Department

Subject

Name Beta Chip Checkout

Address Test Rig 5

43-648

National® Brand

Computation Notebook

Dennison National Company, Holyoke, MA 01041

CPU board
Dave Stephenson

ECO #1 9/19/84

found while comparing VALID to USPM

CSEL decoders U222-225

have CADDR L0<2&3> on the enables

rather than CADDR L0<3&4>

Fix:

Dave 10/5/84

Applied 2MS clock

traced clock through Delay, F74's

→ No output from: U271 (clock buffer)
on U274-6,12,8
CSLO, HI & WETIME

Porquoi non?

Dave 10/8/84

Using PG501 clock module that time

clock = 2MS ~3V sq wave

WE-TIME (U274-8) ~30 ns ⊓

Dave 10/10/84

2MS clk - check WELO, WEHI

(damn 240's feed through!)

alpha sorted netlist in /u/daves/netlist/
tokamaks_list.pretty

selected extracts of TI data book on table

VDD, WE on ~~chips~~ memory OK

!! Another bug! CSLO, CSHI swapped

at input of F240

10/23 Dave S Checking out Tom C's ECOs

ECO #3 (WE swap) OK

ECO #1 (add a bit shift)

a) S.b. U214.6 → U225.5, U224.5, U223.6, U222.6

is U214.6 → U225.5

nc

U222.6, U223.6, U224.5

b) S.b. U214.5 → U1.3, U224.6, U223.5 OK

is U214.5 → U1.3, U224.6

nc

U223.5

is U214.5 → U1.3, U224.6

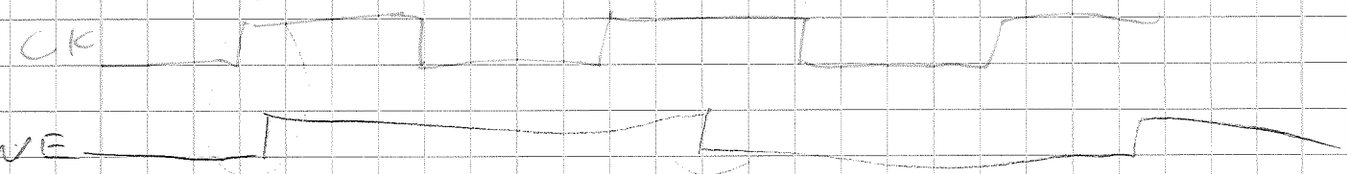
nc also → U222.6, U223.6, U224.5

U223.5

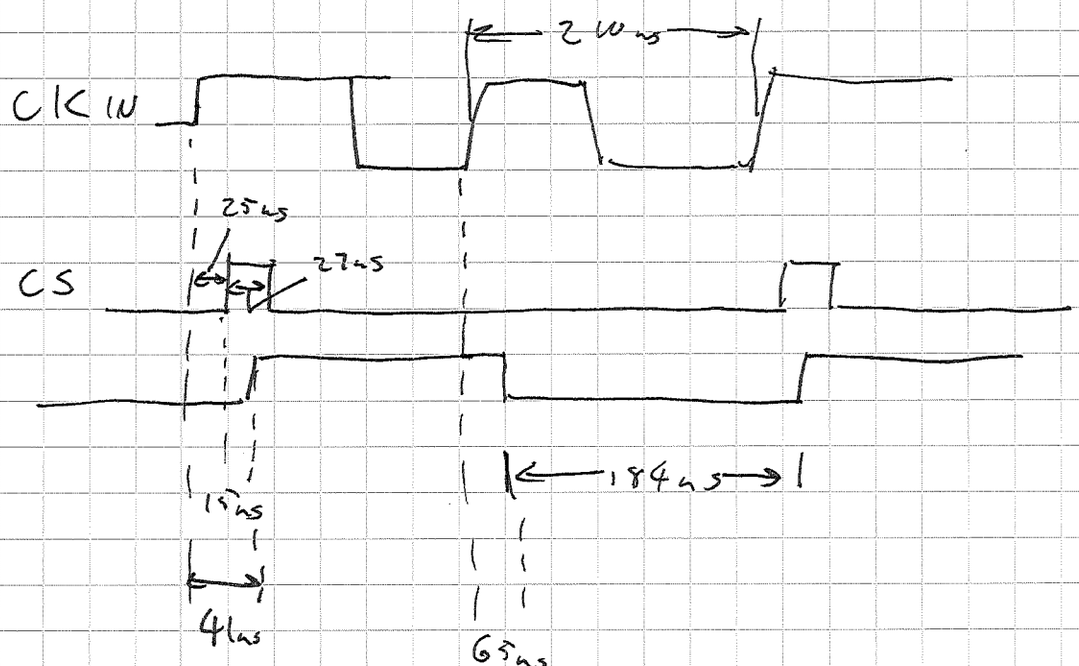
c) S.b. U214.2 → U222.3, U223.3, U224.3, U225.3

is U214.2 → " " " " OK

also → U223.5



CK ÷ 2 = WE — one FF



5983 → 2848

```
 1011 0010 0000
 1011 0101 1111
 1100 0111 1111
```

375
371
307
306
434
178

11/22
11/21
11/23

11/21
11/23
19
9/19
18
13/18
→ 5/18

~~60~~

525, 4976

13

8

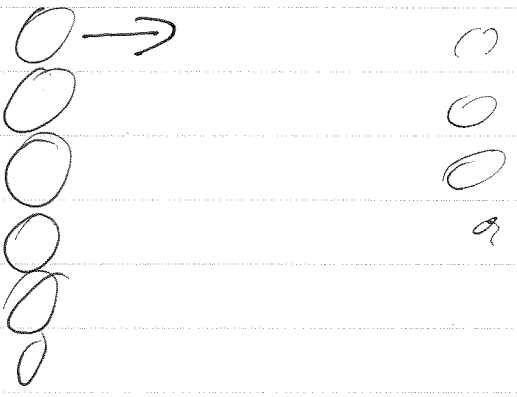
1000001101

1001101111000

10001011.1.1101

~~1307~~

~~3024~~



eg

1 S even

2 S even L odd

3 S odd L Even

4 S odd

```
;;; PHASE 0 —— find send bank with something to send


(defucmacro generate-delivery-phase-0 ()
  (let ((b (delivery-bank-address-length))
        (vp-size *vp-size*)
        (last-vp (* *vp-size* (1- cm:*virtual-to-physical-processor-ratio*))))

    '(ucblock ()
       (label test-send-bank)
       "find send bank with something to send"
       (lls (:a :a) (+ source-vp (constant (eval (delivery-carry-offset))))
            dont-care dont-care sink-flag zero-flag :cond-invert 1)
       "save global to see whether any procs from this bank want to send"
       (ui) (ui (save-global t))

       ,@(when (plusp b)                          ;if VPs
           '((when-not-saved-global
              "if none, go to next send bank"
              (when-less (source-vp (constant ,last-vp))
                (setreg source-vp (+ source-vp (constant ,vp-size)))
                (setreg source (+ source (constant ,vp-size)))
                (setreg cube-address (+ cube-address (constant ,vp-size)))
                (call-delivery-ucode delivery-relativize-address)
                (jump test-send-bank))))
         )
       )))

(def-delivery-ucode delivery-phase-0
  (generate-delivery-phase-0))

;;(declare-delayed-assembly 'delivery-phase-0  :mmcalls '(delivery-relativize-address))

;;; PHASE 1  —— receive end of message while sending msg-p and address


(defucmacro generate-delivery-phase-1 (logior-in-memory)
  "Generates PHASE-1 of the delivery cycle; either ORs incoming message
   with memory or overwrites depending on LOGIOR-IN-MEMORY"

  ;;; these are so common they get short abbreviations
  (let ((c (delivery-chip-cube-dimensions))     ;12 in current hardware
        (b (delivery-bank-address-length))      ;log2 of vp-ratio
        (p (delivery-processor-address-length)) ;4 in current hardware
        (alu-sum (if logior-in-memory
                     '(or :a :b)  ;; :f changed to :b for beta world. -KB0
                     :b)))        ;; here too.

    '(with-mm-reg (relativized-cube-address)
       (setreg relativized-cube-address (constant (eval cm:*start-of-physical-scratch-memory*)))
       "PHASE 1 —— send msg-p and address while receiving message"

       (llsr :doc "send nothing (router sends parity) while receiving"
             :cycle 63
             :r-maddr rbuffer
             :a-maddr temp-dest
             :b-maddr (+ source-vp (constant (eval (delivery-carry-offset))))
             :alu-carry :b  ;;Changed from :f to :b, default is still :f. -KB0
             :alu-sum ,alu-sum
             :flag-write com-e-flag ;;used to be com-d-flag.  -KB0
             :dont-increment-b t)

       (llsr :doc "send send-p again while receiving"
             :cycle 0
             :r-maddr rbuffer
             :a-maddr temp-dest
             :b-maddr (+ source-vp (constant (eval (delivery-carry-offset))))
             :alu-carry :b  ;;Changed from :f to :b, default is still :f. -KB0
             :alu-sum ,alu-sum
             :flag-write com-e-flag ;;used to be com-d-flag. -KB0
             :dont-increment-b t)
```

DONT TOUCH!

RUNNING SYSTEM

DIAGS!

D. Satt

ABEL Version 2.00b Data I/O Corp.

PAL16L8
PAL16L8
BNEXANDH.REV0 4/10/87
TMC CAMBRIDGE MA D. SATTERFIELD
-Translated by TOABEL-

Simulate device P16L8, type 'P16L8'

```
Output                              -> [..........LZZZZHLL.]
V 0013   [101011011-100111000-] ->  [..........HNNNNHHX.]
Test error                          [            ↑    ↑   ]
12 NEXB, H expected
18 NEXC, H expected


Output                              -> [..........LZZZZHLL.]
V 0015   [101011011-100101000-] ->  [..........LNNNNLLX.]
Test error                          [                 ↑   ]
17 NEXD, L expected

Output                              -> [..........LZZZZHLL.]
V 0016   [101011011-100011000-] ->  [..........LNNNNLLX.]
Test error                          [                 ↑   ]
17 NEXD, L expected

Output                              -> [..........LZZZZLLH.]
V 0017   [111111000-101110000-] ->  [..........HNNNNXHH.]
Test error                          [            ↑    ↑   ]
12 NEXB, H expected
18 NEXC, H expected

Output                              -> [..........LZZZZLLH.]
V 0018   [111111000-101100000-] ->  [..........LNNNNXLL.]
Test error                          [                 ↑   ]
19 NEXA, L expected

Output                              -> [..........LZZZZLLH.]
V 0019   [111111000-101010000-] ->  [..........LNNNNXLL.]
Test error                          [                 ↑   ]
19 NEXA, L expected

Output                              -> [..........LZZZZLLH.]
V 0022   [111001000-101000000-] ->  [..........LNNNNXXL.]
Test error                          [                 ↑   ]
19 NEXA, L expected

Output                              -> [..........HZZZZLLL.]
V 0023   [111001000-100100000-] ->  [..........LNNNNXXL.]
Test error                          [            ↑        ]
12 NEXB, L expected
```

seed

→ 7535 to 3856 → 822680

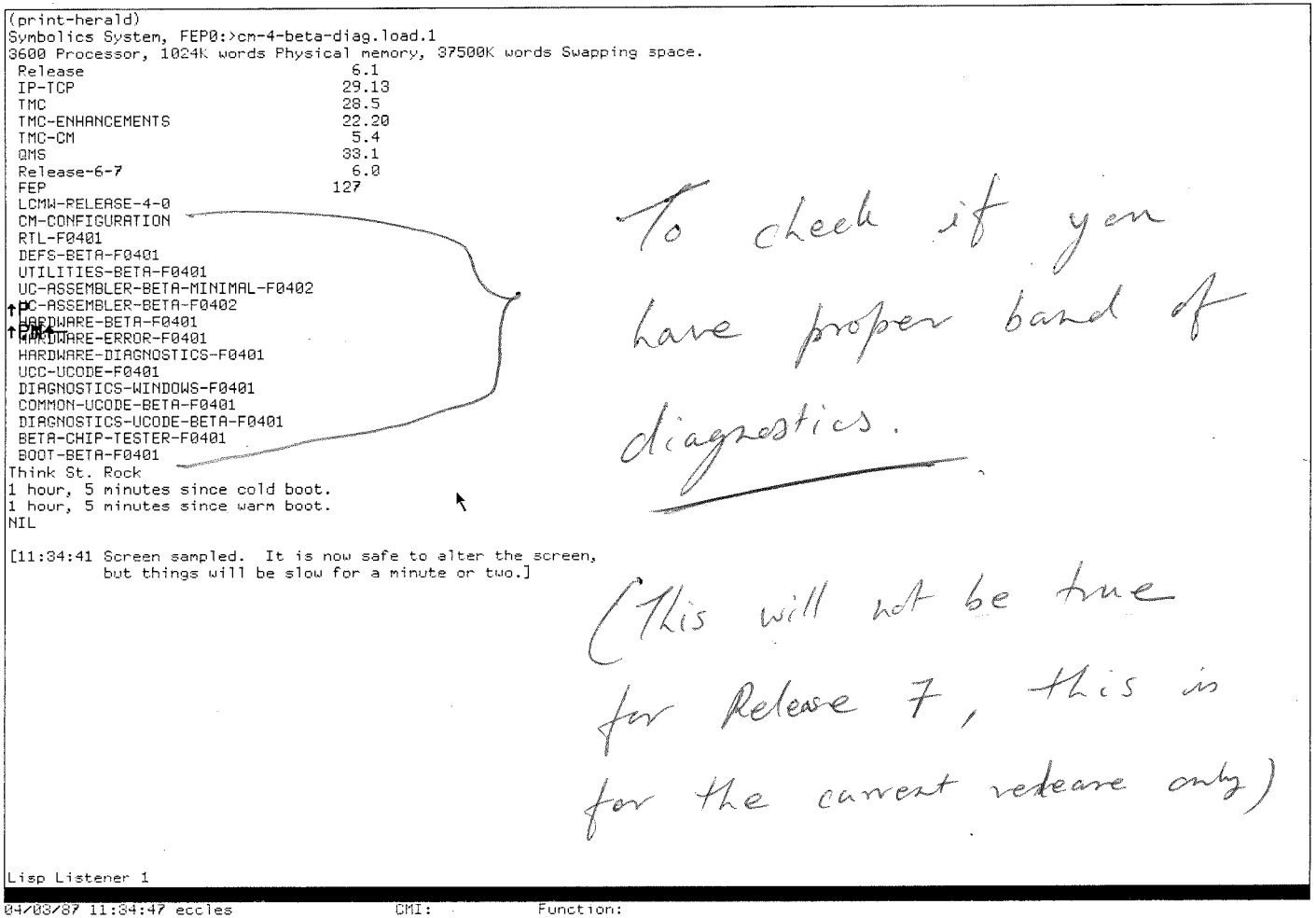
143 to 7520 207304

2207 to 8016, 4426 to 1024

906505

765 to 8128

A1/2 → A3


(print-herald)
Symbolics System, FEP0:>cm-4-beta-diag.load.1
3600 Processor, 1024K words Physical memory, 37500K words Swapping space.
Release 6.1
IP-TCP 29.13
TMC 28.5
TMC-ENHANCEMENTS 22.20
TMC-CM 5.4
QMS 33.1
Release-6-7 6.0
FEP 127
LCMW-RELEASE-4-0
CM-CONFIGURATION
RTL-F0401
DEFS-BETA-F0401
UTILITIES-BETA-F0401
UC-ASSEMBLER-BETA-MINIMAL-F0402
UC-ASSEMBLER-BETA-F0402
HARDWARE-BETA-F0401
HARDWARE-ERROR-F0401
HARDWARE-DIAGNOSTICS-F0401
UCC-UCODE-F0401
DIAGNOSTICS-WINDOWS-F0401
COMMON-UCODE-BETA-F0401
DIAGNOSTICS-UCODE-BETA-F0401
BETA-CHIP-TESTER-F0401
BOOT-BETA-F0401
Think St. Rock
1 hour, 5 minutes since cold boot.
1 hour, 5 minutes since warm boot.
NIL
[11:34:41 Screen sampled. It is now safe to alter the screen,
but things will be slow for a minute or two.]
To check if you have proper band of diagnostics.
(This will not be true for Release 7, this is for the current release only)
Lisp Listener 1
04/08/87 11:34:47 eccles
CMI:
Function:

0/5
2/5
6/5
12/5
12/21
12/29

5/27
9/9
9/11
9/27
13/27

2/19
6/19
12/19
12/23
14/19 ✓

1/2
1/10
1/18
3/18
3/19

4/28
5/28

p6765 p800

TRUST - SYSTEM - CUBE - DIMENSIONS

George

The Connection Machine Model CM-1 Architecture

Brewster A. Kahle and W. Daniel Hillis

Thinking Machines Corporation

Abstract

Massively parallel computer systems offer substantially improved performance over conventional supercomputers for many applications. Designed with many thousands of processing elements operating in parallel, these machines offer a high level of efficiency for computation-intensive algorithms. Machine architectures differ with regard to storage capacities and bandwidths, and these elements, along with software overhead, determine performance. This paper presents the Connection Machine®[1] Model CM-1, a massively parallel computer which exploits data parallelism for its performance. The performance of the CM-1 system is examined with the use of a simple application. Special consideration is given to the effect of problem size and data word size on performance.

1 Introduction

The purpose of this paper is to examine the performance of the Connection Machine Model CM-1 system, a general purpose computer capable of adapting to many sizes and types of data sets. It will provide an overview of the hardware and software systems, and present a simple application from which the raw performance of the machine can be examined. This

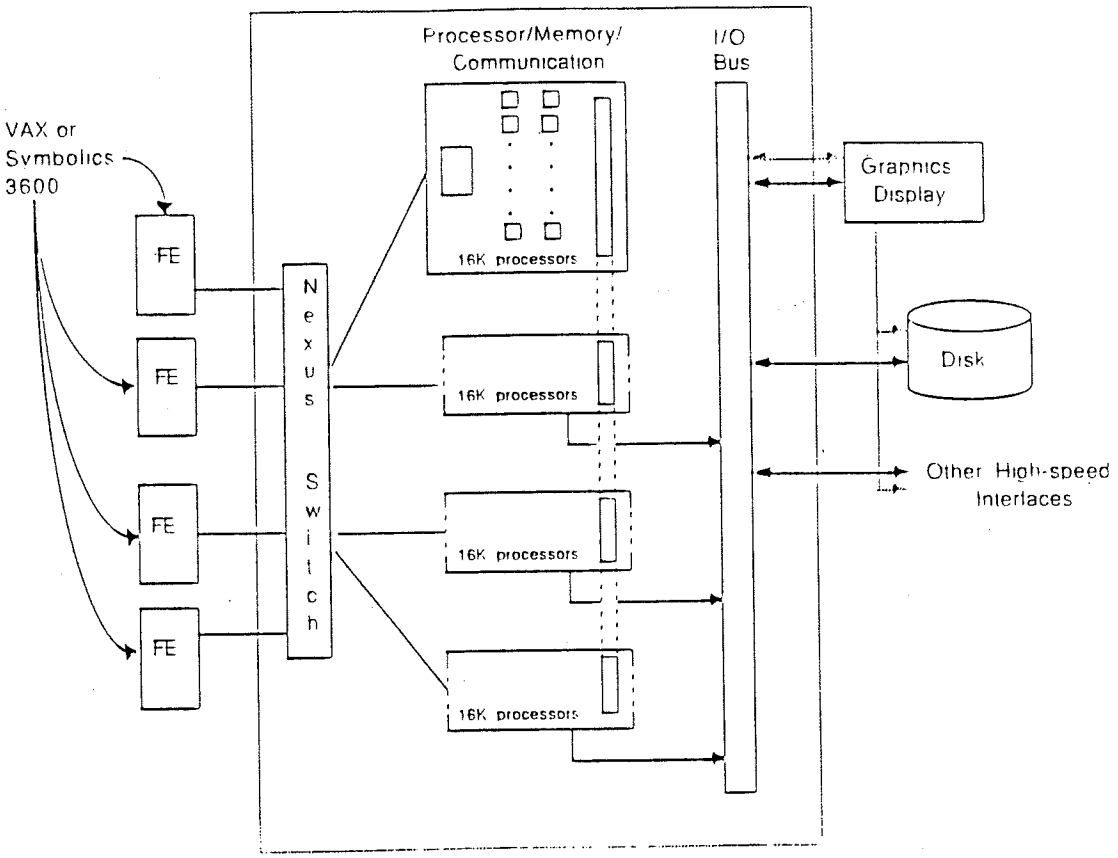
[1] Connection Machine is a registered trademark of Thinking Machines Corporation

will illustrate the power of data level parallelism and indicate how CM-1 performance is enhanced by the use of virtual processors and variable word size.

The Connection Machine system achieves its speed by operating on many thousands of data objects in parallel, rather than looping over them serially. The Connection Machine processors work with a conventional serial front-end computer as a controller, which issues the instructions executed by the Connection Machine computer. This paper describes the first commercial implementation of Connection Machine architecture, the Thinking Machines Corporation Model CM-1.

Other computers have been proposed and built that have varying similarities to the CM-1 architecture. Some machines differ in their control such as the BBN Butterfly[1] and the NYU Ultracomputer[2]. Other machines differ in their inter-communication networks such as the Massively Parallel Processor[6], Non-Von[8], Dado[9], and ICL DAP[10]. Most of the ideas in the CM-1 system have their roots in the long history of both parallel and serial processors that have been proposed and built. The similarities and differences with these machines will not be covered in this paper.

Figure 1: The Connection Machine System



2 Hardware Overview

The CM-1 system consists of a processor array, from one to four front-end computers, and high speed peripherals such as disks and image devices (see Figure 1). The processor array contains 64K processing elements, each a simple serial processor with 4K bits of memory giving 32 MBytes of total memory for the machine. A full CM-1 system is made up of four *sections* of 16K processors each. The sections can be used separately, in pairs, or as one 64K processor unit.

Aside from its processors, each section contains inter-processor communication hardware,

a *sequencer*, and, optionally, a set of peripherals. The inter-processor commumication hardware is used for communication between any processors controlled by one host. A sequencer receives macro-instructions from the front end and broadcasts sequences of micro-instructions to all the processors in its section. A typical macro-instruction would be to add two 32-bit numbers and store them in a particular location. Because of the simplicity of the processors, each macro-instruction is typically implemented by many micro-instructions.

The front end computer, or host, attaches to the microcontroller through a bidirectional crossbar switch called a *Nexus*, and control one, two, or four 16K-processor sections. Typical front ends are a VAX®[2] or a Symbolics 3600®[3] Series computer.

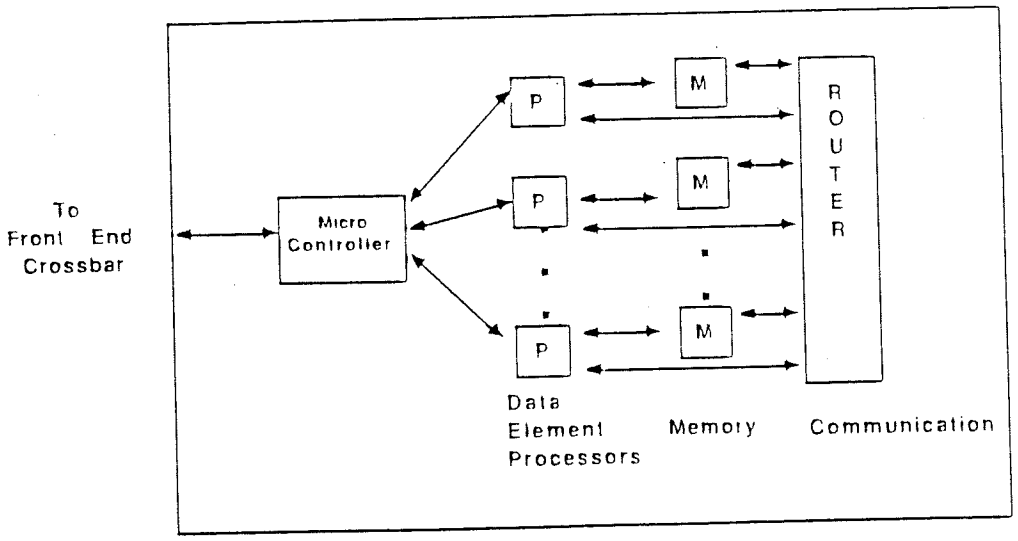
Although the front end can be used to supply data to the CM-1 computer, in many applications the CM-1 processors can process data much faster than the front end can supply it. For this reason, the CM-1 processors are connected to a high speed bidirectional bus consisting of up to 8 I/O channels, each with a bandwidth of 320Mbits/sec. Special disk drives, frame buffers, frame grabbers, and specialized I/O devices can effectively use this bandwidth. This large I/O bandwidth makes the machine effective in very large, data-intensive problems such as databases, image analysis and graphics.

[2]VAX is a registered trademark of Digital Equipment Corporation

[3]Symbolics 3600 is a registered trademark of Symbolics, Inc.

Figure 2

Processor/Memory/Communication System



Pointer referencing (or referencing the data in one data object from another) on the CM-1 computer requires inter-processor communication. Since a CM-1 processor needs to be able to access data from any other processor, this inter-communication system has to handle a large load at high speed. This is achieved by use of a router which is integrated into the architecture so that every processor's memory is easily accessible to every other processor. The result is that the application programmer does not have to worry about physical processor geometry since the router handles all inter-processor communication efficiently regardless of layout.

Along with general pointer referencing, 2-dimensional inter-processor communication is supported. In this type of communication, each data object can communicate with its 2-dimensional neighbor (to the north, east, west, or south). This intercommunication is handled by a slightly different mechanism, and is faster than using the general communication mechanism. Image processing applications often use this ability to move data from one pixel to the next.

Performance of these individual elements will be treated later in this paper.

3 Software Model

From a programming point of view, the CM-1 system can be viewed as a smart memory that can be addressed and controlled by a front end computer. The control of the program lies solely in the front end, while data is distributed between the front end and the CM-1 computer, depending on where it can be operated on most efficiently.

This section is intended as an introduction to programming the CM-1 computer. It will describe how data maps onto the CM-1 processors, and the types of operations that can be performed on CM-1 data. Finally, this section will present an example application and show how it can be implemented on the CM-1 system.

3.1 The CM-1 Computer as Smart Memory

At the heart of any large problem is the data set consisting of some combination of interconnected data objects such as numbers, characters, records, structures, and arrays. In any application this data must be selected, combined and operated on. Data parallelism takes advantage of the parallelism inherent in large data sets.

To make effective use of data parallelism, the CM-1 computer has to be made to "look" like the data of the problem that is being solved. In order to do this, the processors in the machine are each assigned one data object, which can be any combination of immediate data and pointers. Operations can be specified to operate simultaneously on any or all data objects in the machine.

The CM-1 processors should be used whenever an operation can be performed simultaneously on many data objects. Data objects are left in the CM-1 computer during execution of the program, and are operated on in parallel at the command of the front end. This differs from the serial model of reading each data object from its memory one at a time, operating on it, and then re-storing it.

The front end computer handles the flow of control, storage and execution of the program,

and all interaction with the user and/or programmer. The data set, for the most part, is stored on the CM-1 computer. The front end can fetch and store memory in individual processors effeciently to perform serial operations if this is desirable.

There are several direct benefits to maintaining program control only on the front end. First, programmers can work in an environment which is familiar to them. Programming languages, programming concepts, and program development support (compilers, debuggers, editors, etc) do not change. Second, most of the existing code for an application does not have to be changed to use the CM-1 processors. A large part of most application programs pertains to the interface between the program, the user, and the operating system. Since the control of the program remains on the front end, code developed for these purposes is useful with or without the CM-1 computer, and only the code which pertains specifically to the data residing on the Connection Machine computer needs to be coded to use the CM-1 processors. Parts of the program which are especially suited for the front end, such as file manipulation, user interface, and serial data operations, can be done on the front end, while the parts of the program which run efficiently on the CM-1, namely the "inner loop" which operates on the data set, can be done there. In this way, the individual strengths of both the serial front end and Connection Machine computer can be maximized.

Most large problems have a large data set composed of interconnected structures, arrays, and records which can be mapped directly onto CM-1 processors. *Each processor's memory in the CM-1 computer holds the data for one record or array element and some small stack space for temporary results.* A processor can hold data of different types such as integers, floating point numbers, symbols, or pointers. Typically, each processor only needs 128 to 1024 bits for this information. Rarely is running out of memory with one processor a problem because each physical processor contains 4K bits of memory.

Problems of different sizes will use different numbers of processors. Some problems will allocate and de-allocate processors during the course of running the program. Because of the CM-1's flexibility this works well. If a program needs fewer than 64K processors, then some sit idle, ignoring the instructions that are being sent by the front end. If a program needs more assigned processors than 64K, then many more "virtual processors" are available, (see Section 6, Virtual Processor Concept).

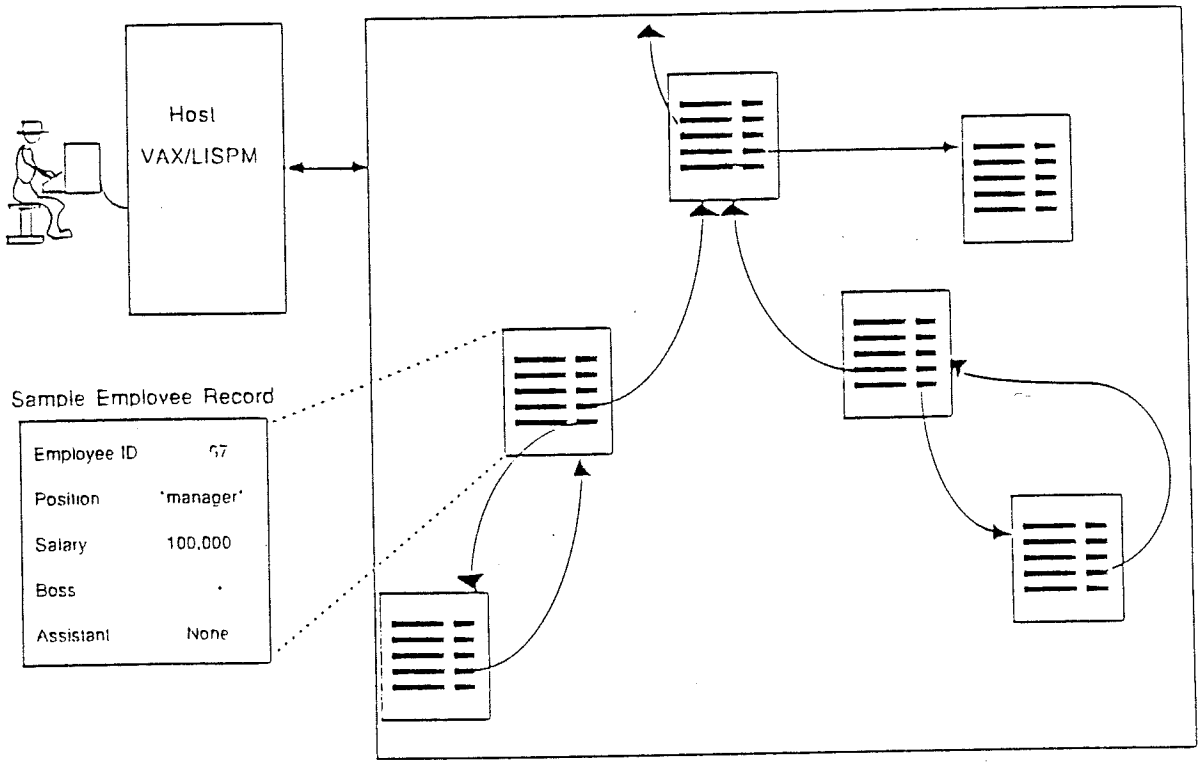
Examples of data objects used in the CM-1 computer are pixels, transistors, database records, atoms, sections of physical space. Each of these are allocated one per processor.

Throughout this paper the terms "processor", "data object" and "record", are used interchangeably.

3.2 Program Example

Consider the problem of manipulating a database of employee records where each employee is a structure of employee ID, title, salary, a pointer to their boss, and a pointer to their assistant (if they have one). The example operations are "give each manager that earns over $100,000 a pay cut of $5,000", "give all managers' assistants a 10% raise" and "sum all the managers' salaries" to illustrate selection and parallel operation, pointer reference, and global operations respectively.

Figure 3
Connection Machine Processors: Software View



For this sample problem, each employee's record would be assigned to a processor. On a serial machine data is loaded from a file and put into records in the virtual memory of the serial machine. On the CM-1 system, the difference is the records are kept on the CM-1 computer.

The language *Lisp[4] (pronounced star-lisp) is used in this paper to describe the pieces of code for the CM-1 computer. All examples can run on the CM-1 computer. In *Lisp the employee structure definition might look like:

```
(*defstruct (employee))
  employee-ID    ;type: number
  position       ;type: number
  salary         ;type: number
  boss           ;type: pointer
  assistant)     ;type: pointer
```

This declaration indicates that structures of this type should be created and manipulated in the CM-1 processors. If the creation and manipulation were done with the serial front end, then these operations would take about as long using the CM-1 processors as using the serial machine's own virtual memory. Thus using the CM-1 memory from the serial machine as if it were its own is not a very significant speed penalty even if one never used the Connection Machine processors.

The data in each processor (in this case, an employee's record) will be a set of fields of numbers and pointers. That a pointer is an address of another processor just means that the program can get or send data to that other processor; this is called moving data through a pointer, or pointer reference. So far nothing is different in the way the CM-1 computer feels or performs from programming the front end. In fact most programs look very much the same whether the data is on the CM-1 computer or not (see Figure 3).

Consider the first operation, to select all managers that earn over $100,000 and decrease their salary $5,000. To execute this we must select all the data objects that satisfy the predicate "is a manager and makes over $100,000". This selection is done by instructing each processor to be "active" when the value in the employee-position slot is equal to

[4]*Lisp is a trademark of Thinking Machines Corporation

manager and the value in the employee-salary slot is greater than $100,000. Notice that *manager* is a value known by the front end and is broadcast to each processor to compare against its local data. In *Lisp the front end code might look like this:

```
(*when (and (=!! position (!! *manager*))
            (>!! employee-salary 100000)))
  (*set employee-salary (-!! employee-salary (!! 5000)))
```

Executing this code on the front end would deliver an instruction stream that would look something like:

```
(let ((saved-context-bit *stack-index*))
  (CM:store-context-always saved-context-bit)
  (CM:u=constant (pvar-location position) *manager* 8)
  (CM:and-context-with-test)
  (CM:u>constant (pvar-location employee-salary) 100000 20)
  (CM:and-context-with-test)
  (CM:u-constant (pvar-location employee-salary) 5000 20)
  (CM:load-context-always saved-context-bit)
  )
```

Notice that these instructions look like a serial computer's macro-instruction set. We have selected a set of processors and executed an operation within those processors.

To perform the "give each managers' assistant a 10% raise" instruction requires communication between processors. This is done by referencing data in the processor pointed to by a field in another processor. In other words, a processor would have an address (pointer) in one of its slots to another processor.

This operation is done in the following steps:

- each manager's processor gets the salary information from their assistant (in the program this is "pref!!"; or parallel reference)

- the manager's processor saves that information in a temporary location. This is a move with the processor to the stack (the "*let")
- the new salary is computed by multiplying the old one by 1.10 in each manager's slot (the "*!!")
- the result is sent back to the assistant's processor and it's put in the salary slot. This involves another move through a pointer. (This is the "*pset", or memory parallel set)

  This operation is done, in *Lisp, as

```
(*let ((assistants-salary!! (pref!! assistant employee-salary)))
   (*pset assistant employee-salary
          (*!! assistant-salary!! (!! .10))))
```

The operation, "total all managers' salary", returns a number to the front end machine which is the total of all managers' salary slots. This is done by selecting all managers and then doing a *global sum*. Global sum is implemented using a logarithmic time tree reduction using the inter-communication wires directly. In *Lisp this looks like:

```
(*when (=!! position (!! *manager*)) ;select managers
  (*sum employee-salary))            ;returns to the front end the total
                                     ;of all processors employee-salaries.
```

This is an example of the use of *scan* that is discussed in the next section.

This example shows how a specific example might map onto the Connection Machine processors and how common operations are used on such data.

4 Basic Performance of the CM-1 System

The performance of the CM-1 system depends on the relative speeds of different common operations. Each application uses a different mix of instructions, so exact

application performance depends on processor utilization and the weighted average of the instructions used. Performance rates shown in Table 1 describe 32-bit operations from the point of view of an active processor as a ratio to the speed of an *add*. Further, this section describes how these numbers scale with different arithmetic precisions and "virtual processor" ratios.

Performance per processor is the operation speed of an active processor or data object. A 32-bit integer addition, for instance, takes about 32 microseconds per processor. The system performance depends on both the speed per processor and the number of active processors. Thus the peak system performance in additions is about 2000 MIPS (million instructions per second). Other elements which impact on performance are the instruction mix used by the application, the precision of the arithmetic and the total number of processors used. Some applications, such as image processing and cellular automata, run significantly faster than 2000 MIPS because they commonly use instructions on less than 32 bits. Table 1 uses 32 bits as a unit of measurement for other performances.

Table 1

Relative Times of Various 32-Bit Operations

Add	1
Move within Processor	.6
Multiply	32
Move with nearest grid neighbor	4
Move through pointer (pointer reference)	25
I/O move (out or into the CM-1)	20
Scan	25
Global max	2
Global sum	25

Table 1 describes the machine as a set of ratios with ADD time. Each entry is described here:

- *The 32-bit add* is the unit in Table 1. An add is a typical logical operation where the operands and results stay within each processor. Each of these fields might be on the stack or a slot in the data object. Other operations having similar speeds are logical operations such as OR, AND, XOR, etc. Also, if one of the operands to these instructions is an "immediate" number (a literal number that comes from the instruction stream rather than from local processor memory) the time is about the same.
- *Move within a processor.* This command moves a field from one location to another within each processor. An example of this is moving an argument to the stack.
- *Multiply* is implemented with a "shift-add" for each bit, so it takes 32 times longer than each add because it is a 32-bit multiply. This corresponds to the *!! in the program example.
- *Move with nearest grid neighbor* are transfers of 32 bits with the processor to the "east", "west", "north" and "south". All active processors will do this transfer at once. This operation is useful in image processing and graphics applications.
- *Move through a pointer* is a reference into another processor's memory. This corresponds to the pref!! (get the assistants' salaries) in the program example. The performance of this operation varies only slightly with the layout of the processors. Of course, if the problem is a grid problem, then the grid communications should be used instead.
- *I/O move* is the time required to get to the I/O port on the CM-1 computer. The real I/O performance will depend on the I/O device being used because most will not sustain a 320Mbit/sec/8K processor rate. The important aspect in this performance is its balance with the other performances so that the CM-1 computer can be connected directly into data-intensive applications.
- *Scan* is an operation that performs a form of processor intercommunication very quickly. Add-scan, for instance, results in each processor getting the sum in all processors before it. This is implemented with a parallel algorithm. Max-scan leaves the maximum up to that point.

Processor #	0	1	2	3	4	5	6	7	8	9	10
Value	0	1	1	0	0	1	1	2	1	0	1
Result of Add-Scan	0	1	2	2	2	3	4	6	7	7	8
Result of Max-Scan	0	1	1	1	1	1	1	2	2	2	2

This is a useful operation on the CM-1 computer for implementing other instructions such as global-sum but many uses have little analogy with serial software cliches.

- *Global-max* results in the maximum value held by any active processor being passed back to the front end computer. The speed of this instruction is due to special hardware. Executing an `AND` of a field in all processors uses the same mechanism and runs at approximately the same speed. Similarly, `OR` and `MINIMUM` perform about the same.
- *Global-sum* is the sum of all values held by processors in the CM-1 computer. This is implemented using *add-scan* and taking the highest processor's result.

5 Performances with Different Word Lengths

The effective performance will change based on what precision arithmetic is needed. "Word Length" is used to describe the precision of the arguments to the CM-1 instructions. The previous section assumed 32-bit arithmetic for each of the operations. The Connection Machine system can operate on a few bits up to hundreds of bits with the same ease of programming. Variable length arguments are handled in microcode for efficiency and ease of programming. The language *Lisp, for example, can extend the word length of numbers as needed. Sometimes a program will need only 16-bit numbers to start, but will use 32 or 64 bit numbers later (or 53 bits for that matter). It is, of course, desirable to minimize the precision required to maximize performance and minimize space utilization.

As one can see in Table 2, most operations scale linearly with the word length. Thus if only 16 bit additions are needed, twice as many can be done in the time of a 32-bit addition. There is a small amount of calling overhead in each instruction call so behavior is not quite linear for very short numbers.

Table 2

Relative Times for Operations on N-bit Numbers

Add	N
Move within Processor	N
Multiply	N^2
Move with nearest grid neighbor	N
Move through pointer (pointer reference)	N
I/O move (out or into the CM-1)	N
Scan	N
Global max	N
Global sum	N

Multiplication uses a shift-add algorithm so that the precision scales with the square of the number of bits; thus a 16 bit multiply would take 1/4 the time of a 32-bit multiply.

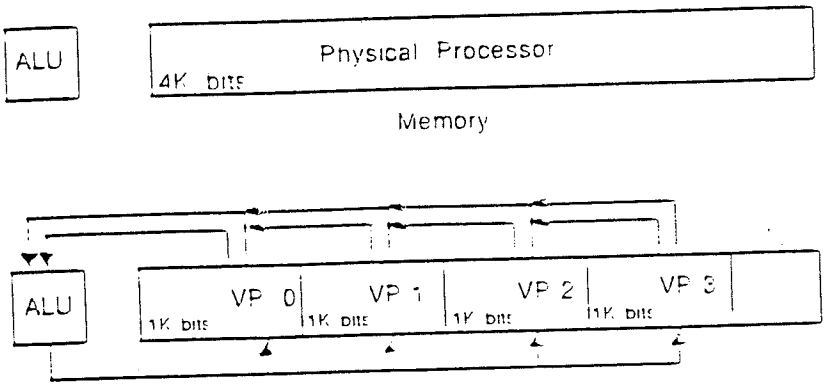
The CM-1 system's flexibility extends to the precision of the arithmetic needed. Image processing applications often use very limited precision numbers, and numerical simulations often use very large precision numbers.

6 Virtual Processor Concept

As presented so far, the CM-1 system has a constant (independent of problem size) performance for an add operation for problem sizes up to 64K processors. In real applications, often 64K processors are not enough. For example, an image processing application with a 1000 by 1000 image would require 1 million processors, one for each pixel. In order to get around the arbitrary limit of the number of physical processors, the CM-1 supports *virtual processors*. Virtual processors are a software abstraction, implemented at the microcode level, which allows a programmer to write programs which are independent of the number of physical processors that the CM-1 hardware contains. This section will describe how virtual processors are implemented on the CM-1, and what effect they have on program performance.

Figure 4

Virtual Processor (VP) Concept



Virtual processors are implemented using three separate mechanisms; one for storage, one for processing, and another for communications. First, the memory of each physical processor is divided evenly among the virtual processors assigned to it. The number of virtual processors per physical processor is referred to as the *virtual processor ratio*. For example, a problem requiring 1M (2^{20}) virtual processors would need a virtual processor ratio of 16 to operate on a 64K machine ($2^{20}/2^{16} = 2^4 = 16$). This means that the physical address space of each virtual processor would be divided into 16 equal address spaces, one for each virtual processor assigned to it. Image processing applications commonly use only 100 bits per virtual processor so

32 processors can be assigned to each physical processor, and the CM-1 computer can be configured as a 2 million processor machine.

The second mechanism which is necessary to support virtual processors is time-multiplexing of the physical processors among the virtual processors assigned to it. Every macro-instruction that is sent by the front end is run on each of the virtual processors within each physical processor. The overhead for switching context is extremely small (about the time to execute a 2-bit add) because each processor is so simple. In the 1M processor example given above, each macro instruction would be executed 16 times by each physical processor.

The third mechanism of communications allow the CM-1 processors to communicate with one another without regard to virtual processors. Grid communications are handled by the microcode by sharing the grid wires. General communications (pointer reference) is handled by the router hardware. The length of a processor address changes based on the number of virtual processors in the machine. This virtual address is used by the router hardware to deliver messages to the correct virtual processor.

Since the virtual processor abstraction is implemented at the microcode level, the virtual processing mechanism is both efficient and totally transparent to the programmer. These characteristics make the CM-1 system appropriate for problems of very different sizes. In general, it is not necessary to know how many physical processors are in a CM-1 computer.

6.1 Effect of Virtual Processors on Performance

Since a physical processor is shared among several virtual processors, the performance from a virtual processor's point of view goes down. If one measures the performance of the CM-1 computer as a whole, however, in terms of MIPs or MFLOPs, the performance does not go down but remains constant. In other words, given a virtual processor ratio of 16, each virtual processor is only getting 1/16 of the physical processors time, so that each macro-instruction takes 16 times as long, but each macro-instruction is doing 16 times as much work as in the case of a virtual processor ratio of 1.

Table 3 shows the effects of virtual processors on the performance of the various classes of macro-instructions. As expected, most operations slow down linearly with

Table 3

Relative Time of Operation with K Virtual Processors

Add	K
Move within Processor	K
Multiply	K
Move with nearest grid neighbor	K
Move through pointer (pointer reference)	KlogK
I/O move (out or into the CM-1)	K
Scan	K
Global max	K
Global sum	K.

the virtual processor ratio. The only exceptions are those that involve interprocessor communication which slow down by $O(Klog(K))$ for virtual processor ratios less than 16. The performance for very high ratios has not been studied. The addition of the $log(K)$ term is due to the increase in router congestion.

Virtual processors free the programmer from the physical size of the CM-1, and also from the size of the data set that the program is to operate on. Since this is done at a very low cost in speed and programming, virtual processors have been used in most applications programmed for the CM-1 system.

7 Conclusion

The Model CM-1 system offers a high level of performance due to its flexible, massively parallel architecture. Much of the maximum peak performance of 2000 MIPS

can be used on a problem due to the balanced performances of all common instructions. The CM-1 can be used for very different types of applications because the machine can be efficiently reconfigured to have the needed number of processors and the needed arithmetic precision.

Since the CM-1 computer works closely with a serial front end computer, the programmer operates in a familiar environment. The CM-1 system combines the benefits of programming the front end computer with the performance of a massively parallel machine.

References

[1] Bolt Beranek and Newman Inc. "Development of a Butterfly Multiprocessor Test Bed," Report No. 5872, Quarterly Technical Report No. 1, March 1985.

[2] Gottlieb, Allan, Ralph Grishman, Clyde P. Kruskal, Kevin P. McAuliffe, Larry Rudolph, and Marc Snir. "The NYU Ultracomputer – Designing an MIMD Shared Memory Parallel Computer," *IEEE Transactions on Computers*, Vol. C-32, No. 2, February 1983, pp. 175-189.

[3] Hillis, W. Daniel, *The Connection Machine*, The MIT Press, Cambridge, MA 1985

[4] Introduction to Data Level Parallelism, Thinking Machines Technical Report, 86.14, April 1986

[5] Lasser, C. and S. Omohundro, "The Essential *Lisp Manual", Thinking Machines Corporation, July 1986

[6] Schaefer, D. H., J. R. Fischer, and K. R. Wallgren. "The Massively Parallel Processor," *Engineering Notes*, Vol. 5, No. 3, pp. 313-315, May-June 1982. (More MPP)

[7] Siegel, Howard Jay, Leah J. Siegel, Frederick C. Kemmerer, Philip T. Mueller, Jr., Harold E. Smalley, Jr., and S. Diane Smith. "PASM: A Partitionable SIMD/MIMD System for Image Processing and Pattern Recognition," *IEEE Transactions on Computers*, Vol C-30, No. 12, December 1981.

[8] Shaw, David Elliot. "The NON-VON Supercomputer," Department of Computer Science, Columbia University, August 1982.

[9] Stolfo, Salvatore J. and David Elliot Shaw. "DADO: A Tree-Structured Machine Architecture for Production Systems," Department of Computer Science, Columbia University, March, 1982.

[10] Flanders, P.M., et. al. "Efficient High Speed Computing with the Distributed Array Processor," *High Speed Computer and Algorithm Organization*, Kuch, Lawrie, and Sameh, Eds. Academic Press, New York, 1977, pp. 113-127.

Appendix E: Design Review (12/29/86) Minutes

The minutes of the micro controller schematic design review follow.
We would like to thank all those that were able to attend for their input.

1. Does the IFIFO and OFIFO need all the flags (4 each) combined? Yes, it is done this way in order to handle the asynchronous nature of the parts.

2. Several signal names to/from the Nexus have been renamed to make make their meaning more obvious. It was suggested that perhaps we didn't want to do this because of their difference from the alpha machine. Signal names will be changed only when it helps to clarify their meaning. All signal names that are changed need to be changed on all documentation (i.e Nexus).

3. All unused inputs should be pulled either high or low. This will keep the ICs from oscillating and will enable ATE of the board. Good design practice will be used in doing this to adequately support ATE requirements.

4. The method of supporting a micro code parity error was discussed at length.
   a. If one of the four micros has a parity error, all will trap but the one that had the error will trap one instruction sooner than the others.
   b. Software will not use the micro word parity error trap to signal software detected error conditions. A special UC trap condition has been provided in encoded field A for this purpose.
   c. The error handler should be smart enough to be able to pick out the failed bit in the micro word on a legitimate parity error. This will require the error handler to read the valid word from the control store of the three micros that didn't fail.

5. The method of reclocking the refresh signal (sheet 8) was questioned. This will be looked at again.

6. Timing diagrams are needed for clock (and other) signal, both local to the micro controller and globally in the system. This need is recognized and will be addressed as time permits.

7. Should series resistors be included in the data lines of the scratch ram and the control store? Perhaps, but they will not be used in the wire wrap version. If testing of the board indicates that they are needed, they will be included in the multi-wire (PC) board. Series resistors will, however, be added to the address lines of the scratch ram.

8. It was suggested that space be left to support at least 8 extra bits in the micro word. This will be done, provided that there is sufficient real estate available.

9. The signal names TRAP OE L and RGND L on sheet 14 seem to be wrong. In fact, they are correct. The meaning of TRAP OE L is to enable the trap vector to the BR BUS. It is also used to clear the trap bits that the host can set in status register 0. For this reason, the name looks misleading.

10. The control of the performance monitor was discussed (again). There still seems to be no consensus on the desired way to implement this control so it will remain as defined in the micro word definition.

11. Does the sequencer carry in signal need to be controlled by the micro code or can it be tied high? No additional instructions can be added by giving the micro code control over this signal without a significant amount more logic. The alpha machine drives this bit low only when running from prom and not actively loading the control store (i.e. waiting for data to load).

12. It was suggested that the TI part (74ALS2240 or 74ALS2540, 2541), incorporating series resistors, be used to drive the address lines of the control store. This would use less real estate than 74F244's and separate resistors. The new parts will not be used because they have lower drive current and are slower than the 74F244's.

13. Perhaps the YDEST pal should be changed to a registered device and clocked with the falling edge of the clock (middle of instruction). This would guarantee clock enable hold time on Y Bus destination devices. Yes, this will be done.

14. A better understanding is needed of the INSTR timing during reads. It is believed that the current hardware will support the needs of the matrix board. As we get to understand the timing better (when it's documented)

MICRO-INSTRUCTION

-Mahesh Ganmukhi, Jim Lalone

This document presents the micro-instruction definition for the Beta Machine Microcontroller. The exact hardware implementation and detail description of various functional blocks is not provided in this document.

**********LOG OF CHANGES**********

11/11/86 Mahesh: Created.

11/20/86 Mahesh: Changes/suggestions from the design review (11/14/86) incorporated.

12/05/86 Mahesh: Refinements, added bit locations.

12/19/86 Jim : Added section on status reg., traps and WCS download.

1/02/87 Frye: Fixed spelling errors.
Corrected bit position for LIT

1/21/87 Mahesh: Changed UW BR CNTL <3:0> field and added explanation to DBUS SRC SEL <3:0> field.

2/28/87 Mahesh: Added correction to the WEHI WELO (see WE CNTL field), note on loading the OFIFO (see YBUS DEST SEL field), note on Maddr upper bound compare (see LD MADDR field), change in ENC FLD A field and ENC FLD B field and ALU RAM SHIFT (see BR CNTL field and also RESTRICTIONS and APPLICATION NOTES.

**********END OF THE LOG**********

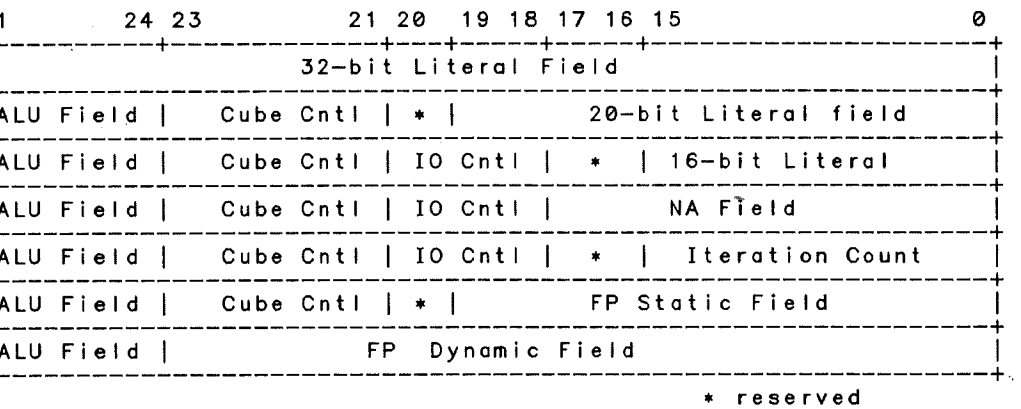
1) UW SEQ I <3:0>

This four bit field is used to specify an instruction to the sequencer (IDT49C410A). The sequencer is functionally equivalent to the 2910. There are two differences between this part and the 2910. All the data paths inside this sequencer are 16 bit wide and the stack is 33 locations deep.

This field is also used to control the logic for generating the most significant two bits of the micro-PC.

2) UW LIT <31:0>

This is a 32-bit literal field. This field is used for sourcing various fields as shown below.



2.1) A 32-bit literal field (LIT <31:0>) sourced on the D-bus.

2.2) A 20-bit literal field (LIT <19:0>) sourced on the D-bus, Cube Control (LIT <23:21>) and ALU field (LIT <31:24>).

2.3) A 16-bit literal field (LIT <15:0>) sourced on the D-bus, IO Control (LIT <20:18>), Cube Control (LIT <23:21>) and

ALU field (LIT <31:24>).

2.4) The next address field (LIT <17:0>) sourced on the BR-bus, IO Control (LIT <20:18>), Cube Control (LIT <23:21>) and ALU field (LIT <31:24>).

2.5) Input to the iteration counter inside the 2910 sequencer (LIT <15:0>) sourced on the BR-bus, IO Control (LIT <20:18>), Cube Control (LIT <23:21>) and ALU field (LIT <31:24>).

2.6) Floating Point Static field (LIT <19:0>), Cube Control (LIT <23:21>) and ALU field (LIT <31:21>).

2.7) Floating Point Dynamic Field (LIT <23:0>) and ALU field (LIT <31:24>).

Following is a description of the various fields mentioned above.

2.a) CUBE CNTL <2:0>

This field controls the generation of the SENDREVEN, SENDRODD, LATCHREVN and LATCHRODD signals as shown below. This field was previously known as the ROUTER CONTROL.

The CUBE CNTL <2:0> decode 0, as shown below, is used to specify no operation (NOP). Whenever the CUBE CNTL <2:0> is not specified in the UW LIT <31:0>, as in 2.1 and 2.7 above, the signals generated default to NOP.

CUBE CNTL<2:0>		SENDREVEN	SENDRODD	LATCHREVN	LATCHRODD
(NOP)	0	0	0	0	0
	1	1	0	0	0
	2	1	0	0	1
	3	0	1	1	0
	4	0	1	0	0
	5	1	0	1	0
	6	0	1	0	1
	7	Reserved			

2.b) IO CNTL <2:0>

This field is used for generating I/O control signals as shown below. Note that whenever the IO CNTL <2:0> is not specified in the UW LIT <31:0>, as in 2.1, 2.2, 2.6 and 2.7 above, the signals generated default to NOP.

IO CNTL <2:0>	FUNCTION
0	NOP
1	Read IO Board Reg.
2	Write IO Board Reg.
3	Read Block A
4	Write Block A
5	Read Block B
6	Write Block B
7	reserved

2.c) ALU <7:0>

This is a eight bit literal field. This field, along with Y BUS <11:4> and UW ARG INSTR, is used to generate nanoinstruction INSTR <11:4> in the following way.

```
INSTR <11:4> := !UW ARG INSTR & UW ALU <7:0>
                #
                UW ARG INSTR & (UW ALU <7:0> $ Y bus <11:4>);
```

Where, ! means inverse, & means logical AND, # means logical OR and $ means logical XOR.

2.d) FP DYNAMIC <23:0>

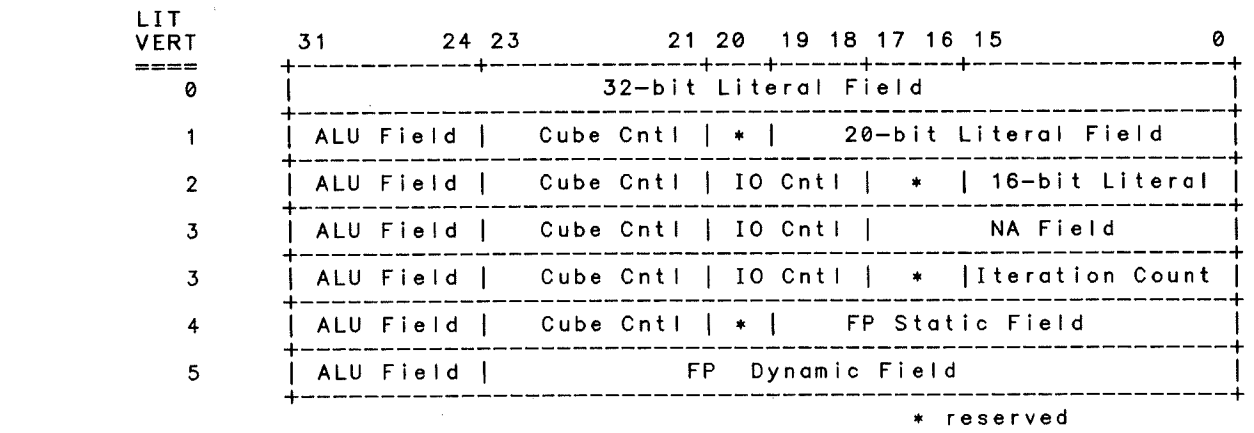
This field is used for sending instruction to Weitek MAC. The control instruction is split into two group of fields. The FP DYNAMIC FIELD can change every cycle, whereas the FP STATIC FIELD is latched in the matrix board and changed very few times. Please refer to Sprint User's Manual for details.

2.e) FP STATIC <19:0>

Please refer to Sprint User's Manual for details.

3) UW LIT VERT <2:0>

This field indicates the presence of one of the six vertically encoded LIT fields as shown below.



Note that selection of UW LIT VERT = 4 will cause the FP STATIC field to be latched in the matrix board. This is done by asserting LD STAT signal. If UW LIT VERT is not equal to 4, the LD STAT signal is not asserted.

Whenever the UW LIT VERT is not equal to 5, STALL signal is asserted. The STALL signal, when asserted, informs the matrix board that the data in the current FP Dynamic Field is not valid data and the current Weitek cycle must be killed.

LIT VERT = 6 and 7 are reserved.

4) UW BR CNTL <3:0>

This is the branch control field. It is used to select 1 of 16 branch conditions to be tested. Following table shows the various branch conditions available.

UW BR CNTL <3:0>		BRANCH CONDITION	
0		* FALSE L	
1		SAVED GLOBAL H	
2		SR <31> H	(Shift Reg. 31)
3		SR <0> H	(Shift Reg. 0)
4		IO READY H	
5		IFOR H	
6		OFIR H	
7		OF EMPTY H	
8		ADDR ZERO H	(ALU F=0)
9		ADDR CO H	(ALU CO)
A		ADDR NEG H	(ALU F31)
B		IFOR & !IO READY H	
C		** RAM31 H	
D		** RAM0 H	
E	reserved		
F	reserved		

* The non-inverted FALSE branch condition may be used as NOP. The inverted FALSE branch condition may be used as unconditonal branch.

(IGNORE)** This branch condition is either RAM31 or RAM0, depending upon
(IGNORE) the direction of the ALU RAM shift.

** Note that RAM31 and RAM0 are two distinct branch conditions.

5) UW INVT BR

This is the invert branch field which allows the inverts of the branch

conditions to be selected by the UW BR CNTL field. When this bit is asserted, an inverted branch condition is selected.

UW INVT BR	FUNCTION
0	non-inverted condition/NOP
1	inverted condition

6) UW BR-BUS SRC SEL

The branch address bus source select field is used to select either the branch register or the NA/TA (Next Address/Trap Address) buffer.

UW BR-BUS SRC SEL	FUNCTION
0	Select NA/TA Buffer/NOP
1	Select Branch Register

Note that occurrence of a trap condition will override source selection defined by this field and the hardware will source the NA/TA buffer on the BR-bus, e.g., if the Dispatch Register is selected as the source on the BR-bus and a trap condition occurs, then the NA/TA buffer will be forced as the source on the BR-bus.

7) UW DP I <9:0>

This is the instruction field of the ALU. The ALU is built using two 16 bit IDT49C402A slices. This part is functionally equivalent to 2901. There are two differences between this part and the 2901. The IDT part contains 64 deep register file and it has an additional destination control bit. The 10 bit instruction field is subdivided into the source operand control, function control and destination control as described below.

7.1) ALU SOURCE OPERAND CONTROL <2:0>

This three bit field specifies one of eight source operand pairs available to the ALU. Following table shows the decode of these three bits and their function.

UW DP I <2:0>	ALU SOURCE OPERAND R	S
0	A	Q
1	A	B
2	0	Q
3	0	B
4	0	A
5	D	A
6	D	Q
7	D	0

7.2) ALU FUNCTION CONTROL <2:0>

This three bit field specifies one of eight functions to the ALU. Following table shows the decode of these three bits and associated ALU function.

UW DP I <5:3>	ALU FUNCTION	
0	R Plus A	
1	S Minus R	
2	R Minus S	
3	R OR S	
4	R AND S	
5	!R AND S	(!R means $\overline{R}$)
6	R EX-OR S	
7	R EX-NOR S	

7.3) ALU DESTINATION CONTROL <3:0>

This four bit field provides sixteen destination control functions. In the following table, first eight are the standard 2901 destination control functions and the following eight are the new destination control functions.

	RAM FUNCTION	Q REG FUNCTION	Y	RAM SHIFTER	Q SHIFTER

UW DP I <9:6>	SHFT	LOAD	SHFT	LOAD	OUTPUT	R0	R15	Q0	Q15
Standard 2901 destination control functions:									
8	x	none	none	F->Q	F	x	x	x	x
9	x	none	x	none	F	x	x	x	x
A	none	F->B	x	none	A	x	x	x	x
B	none	F->B	x	none	F	x	x	x	x
C	down	F/2->B	down	Q/2->Q	F	F0	IN15	Q0	IN15
D	down	F/2->B	x	none	F	F0	IN15	Q0	x
E	up	2F->B	up	2Q->Q	F	IN0	F15	IN0	Q15
F	up	2F->B	x	none	F	IN0	F15	x	Q15
New destination control functions:									
0	none	D->B	none	F->Q	F	x	x	x	x
1	none	D->B	none	F->Q	F	x	x	x	x
2	none	F->B	none	D->Q	F	x	x	x	x
3	none	F->B	none	D->Q	A	x	x	x	x
4	x	none	down	Q/2->Q	F	x	x	Q0	IN15
5	none	D->B	x	none	F	x	x	Q0	x
6	x	none	up	2Q->Q	F	x	x	IN0	Q15
7	x	none	none	D->Q	F	x	x	x	Q15

8) UW DP CN

This is the carry-in bit of the ALU.

9) UW DP A <5:0>

This six bit field selects 1 of 64 registers to be read through the A-port of the register file.

10) UW DP B <5:0>

This six bit field selects 1 of 64 registers to be read through the B-port of the register file. This field also specifies the destination address for a write register.

11) UW DBUS SRC SEL <3:0>

This is the DBUS source select field. This field is used for sourcing various devices on the D-bus and also for controlling the word (16-bit) swapper as shown below. In the following table the word swap means sourcing upper 16 bits <31:16> on lower 16 bits <15:0> and vice versa. Longword means 32 bits.

UW DBUS SRC SEL <3:0>		DBUS SOURCE
0		Input Fifo, Source Longword & word swap
1		Input Fifo, Source Longword & no swap
2	*	Input Fifo, Source upper 12 bits <31:20> on lower 12 bits <11:0>
3	*	MADDR Register
4	# *	Chip Select & IO Select Register
5	$	Rdata Register & word swap
6	*	Rdata Register & no swap
7	*	Scratch RAM address
8	R*	Scratch RAM data & word swap
9	R*	Scratch RAM data & no swap
A		Shift Register
B	**	Literal field
C		Op and FP Register
D		Reserved
E	R*	LSB 32-bit of Performance Monitor
F	R*	MSB 8-bit of Performance Monitor

* Zero extended data.

$ Zero extended before swapping.

** When the literal field is selected as the source on the D-bus, 32, 20 or 16 bit literal is sourced depending upon UW LIT VERT. If the UW LIT VERT is equal to 0, then 32-bits are sourced; if the UW LIT VERT is equal to 1, then zero extended 20-bits <19:0> are sourced and if the UW LIT VERT is equal to 2, then zero extended 16-bits are sourced on the D-bus.

If the literal field is selected as the source on the D-bus and

the UW LIT VERT is not equal to either 0, 1 or 2; then the D-bus data is undefined.

R* Note the restriction in 'RESTRICTIONS and APPLICATIONS NOTES' section.

* Following data will be sourced on the DBUS when DBUS SRC SEL field is 4.

CHIP SEL <12:0>	->	DBUS <12:0>
IO SEL <1:0>	->	DBUS <14:13>
BUF IO SEL <1:0>	->	DBUS <16:15>
BOARD ID EVEN	->	DBUS <17>
BOARD CONFIG <1:0>	->	DBUS <19:18>
BACKPLANE <2:0>	->	DBUS <22:20>
ODD FLAG	->	DBUS <23>

12) UW YBUS SRC SEL

This is the YBUS source select field.

UW YBUS SRC SEL	YBUS SOURCE
0	ALU output/NOP
1	ALU bypass buffer

13) UW YBUS DEST SEL <3:0>

This is the YBUS destination select field.

UW YBUS DEST SEL <3:0>		DESTINATION
0	*	NOP
1	**	Output Fifo, Low word <15:0>
2	**	Output Fifo, HI word <31:16>
3	**	Output Fifo, Longword <31:0>
4		Chip Select Register
5		Dispatch Register
6		Scratch RAM Addr Counter
7		IO Sel Register
8		Shift Register
9	$	Configuration Register
A		Reserved
B		Reserved
C		Reserved
D		Reserved
E	R*	Performance Monitor LS 32 bits
F	R*	Performance Monitor MS 8 bits

* Note that Instruction Register is not provided as an explicit destination. Loading of the Y-bus data into the Instr. Reg. is controlled by the UW ARG INSTR. Therefore, when certain data is sourced on the Y-bus with (probable) destination of the Instr. Reg. NOP must be specified in the Y-bus destination select field.

(IGNORE)** Note that writing to the Output Fifo, Hi Word or the Output Fifo
(IGNORE) Longword causes loading of the data (i.e., push) in the Output Fifo.

** Note that writing to the Output Fifo, Hi Word causes loading of the the data (i.e., push) in the Output Fifo if Low word was loaded previously.
Note that writing to the Output Fifo, Low Word causes loading of the the data (i.e., push) in the Output Fifo if Hi word was loaded previously.
Note that writing to the Output Fifo, Longword causes loading of the the data (i.e., push) in the Output Fifo.

R* Note the restriction in 'RESTRICTIONS and APPLICATIONS NOTES' section.

$ The Configuration Register is an two bit register. This register is used for loading system configuration information, e.g., the memory size (64K, 256K, 1Meg) on the matrix board, etc.

14) UW LD MADDR <1:0>

This two bit field is used for loading the MADDR register, the lower bound comparator and the upper bound comparator.

UW LD MADDR <1:0>	FUNCTION
0	NOP
1	& Load Lower Bound Register
2	&# Load Upper Bound Register
3	* Load MADDR Register

* The 'Load MADDR Register' should only be specified when the CM opcode is not zero (NOP) or when MADDR register is used to address an IO register.

& The upper and lower bound registers are 16 bit registers. The upper and lower bound comparison is done only for most significant 16 bits of the MADDR i.e., for MADDR <19:4>.

Note that upper and lower bound error checking is implemented in the way descibed below.

Error if : Upper Bound <= Maddr or Maddr < Lower Bound

Not Error if : Upper Bound > Maddr and Maddr >= Lower Bound

Therefore, upper 16 locations of 1 Meg memory are not usable since upper bound of FFFF (MADDR <19:4>) will cause upper bound error when MADDR <19:0> is in the range: FFFFF -> FFFF0.

15) UW ARG INSTR

This field controls sourcing of the UW FLAG, UW ALU and UW PMODE (described below) fields into the nanoinstruction field.

16) UW FLAG <3:0>

This is a four bit literal field. This field, along with YBUS <3:0> and UW ARG INSTR, is used to generate instruction INSTR <3:0> in the following way.

```
INSTR <3:0> := !UW ARG INSTR & UW FLAG <3:0>
               #
               UW ARG INSTR & (UW FLAG <3:0> $ Y BUS <3:0>);
```

Where, ! means inverse, & means logical AND, # means logical OR and $ means logical XOR.

17) UW PMODE <3:0>

This is a four bit literal field. (Note that the IFLIP SEL <1:0>, which used to be a field in the alpha microcontroller's microword, is replaced by this PMODE field.)

This field; along with Y BUS <15:12>, UW ARG INSTR, ODD FLAG (output of 'odd flip-flop' used for detecting the odd and even cycles.) and PMODE SEL; is used to generate nanoinstruction INSTR <15:12> in the following way.

```
INSTR <14:12> := !UW ARG INSTR & UW PMODE <2:0>
                 #
                 UW ARG INSTR & (UW PMODE <2:0> $ Y BUS <14:12>);

INSTR <15> := !UW ARG INSTR & UW PMODE <3>
              #
              UW ARG INSTR & OP6 & ODD FLAG
              #
              UW ARG INSTR & !OP6 & (UW PMODE <3> $ Y BUS <15>);
```

means OR
& means AND
$ means XOR

18) UW SRAM W

This bit is used for Scratch RAM read/write control.

UW SRAM W	FUNCTION
0	Scratch RAM Read/NOP
1	Scratch RAM Write

19) UW WE CNTL <1:0>

This field controls the generation of the WE HI and WE LO signals.

WE CNTL <1:0>		ODD FLAG	WE HI	WE LO
(NOP)	0	x	0	0
	1	x	1	1
	2	0	1	0
	2	1	0	1
	3	Reserved		

20) UW OP <5:0>

This is a six bit literal field. This field is used to specify 3-bit op code to the CM chip (UW OP <2:0>) and 3-bit op code to the sprint chip (UW OP <5:3>). Please refer to CM chip (beta) specification and sprint chip specification for details.

21) UW ODD PARITY

This is the odd parity bit associated with a microword.

22) UW XTND CLOCK

This field controls extension of the clock distributed to the CM. This bit, when active, extends the HIGH phase of the clock signal distributed to the CM. Internal clocking of the microcontroller is not affected by this bit.

23) UW BOUND CHK

This field controls the MADDR bound checking function as shown below.

UW BOUND CHK	FUNCTION
0	Check MADDR bounds/NOP
1	Override bounds checking

24) UW ENC FLD A <3:0>

This is a general purpose encoded field. This field can be used to perform various functions shown below.

UW ENC FLD A <3:0>	FUNCTION
0	NOP
1	POP Input FIFO
2	LATCH LED
3	SAVE GLOBAL
4	Set Trap Enable
5	Set Trap Disable
6	Set Page mode - Normal
7	Set Page Mode Fault - Always
8	Set Page Mode Fault _ Never
9	Increment Scratch RAM counter
A	Decrement Scratch RAM counter
B	Reset Odd Flip-flop
C	Toggle Odd Flip-flop
D	Force UC trap
E	Force CM (instruction) parity error
F	

25) UW ENC FLD B <2:0>

This is another general purpose encoded field. This field can be used to perform various functions shown below.

UW ENC FLD B <2:0>	FUNCTION
0	NOP
1	Shift SR
2	$ Force CM (matrix buffers) parity error
3	Set 'Global Out' to input from CM
4	Set 'Global out' to input from SR
5	* CM direct R/W access
6	** Force Output FIFO parity error

7 Force Scratch RAM & IFIFO parity error

* CM direct R/W access is used to access CM memory as a scratch memory. Note that in a multi microcontroller configuration each UC will get valid data.

** Whenever the Output FIFO parity error is forced, the Y-bus destination select field must specify 'Output FIFO, Longword' or 'Output FIFO, Low word'.

$ Force CM (matrix buffers) parity error is used to force parity error on buffers driving nano-insruction to the matrix board, except the INSTR buffers.

26) UW PERF MON

This field controls the performance monitor. The performance monitor is a 40-bit counter. This counter can be loaded from the Y-bus and its data can be sourced on the D-bus.

UW PERF MON	FUNCTION
0	NOP
1	Increment Counter

27) UW BREAKPOINT

This bit can be used to force a breakpoint. Whenever this bit is asserted, the UC data path is set into a 'hold' mode.

This state of the UC is terminated by the HM setting bit 5 (continue) in the status register 0.

UW BREAKPOINT	FUNCTION
0	NOP
1	Force Breakpoint

Total: 109 bits used
3 spares

112

BIT POSITION OF MICRO-WORD

The following is the bit position definition of the micro-word in the static RAM array.

UW FIELD	BIT POSITION		SIGNAL NAME
UW DP I <9:0>	WCS <9:0>		DP I <9:0> H
UW DP CN	WCS <10>		DP CN H
UW DP A <5:0>	WCS <16:11>		DP A <5:0> H
UW DP B <5:0>	WCS <22:17>		DP B <5:0> H
UW YBUS SRC SEL	WCS <23>		YBSRCS L
UW DBUS SRC SEL <3:0>	WCS <27:24>		DBSRCS <3:0> H
UW YBUS DEST SEL <3:0>	WCS <31:28>		YDEST <3:0> H
UW LIT <31:0>	WCS <63:32>		LIT <31:0> H
UW LIT VERT <2:0>	WCS <66:64>		LIT VERT <2:0> H
UW LD MADDR <1:0>	WCS <68:67>		LD MADDR <1:0> H
UW WCS SP <0>	WCS <69>	Spare Bit	
UW BREAK POINT	WCS <70>		BREAK POINT H
UW SRAM W	WCS <71>		SRAM W L
UW FLAG <3:0>	WCS <75:72>		FLAG <3:0> H
UW PMODE <3:0>	WCS <79:76>		PMODE <3:0> H
UW WE CNTL <1:0>	WCS <81:80>		WE CNTL <1:0> H
UW OP <5:0>	WCS <87:82>		XOP <5:0> H
UW ODD PARITY	WCS <88>		ODD PARITY H
UW XTND CLOCK	WCS <89>		XTND CLOCK H
UW UW PERF MON	WCS <90>		INC PM L
UW INVT BR	WCS <91>		INVT BR L
UW BR CNTL <3:0>	WCS <95:92>		BR CNTL <3:0> H
UW SEQ I <3:0>	WCS <99:96>		SEQ I <3:0> H
UW BR-BUS SRC SEL	WCS <100>		BRBSRCS H
UW ENC FLD A <3:0>	WCS <104:101>		ENC FLD A<3:0> H
UW ENC FLD B <2:0>	WCS <107:105>		ENC FLD B<2:0> H
UW BOUND CHK	WCS <108>		BNDCHK L
UW ARG INSTR	WCS <109>		ARG INSTR H
UW WCS SP <1>	WCS <110>	Spare Bit	
UW WCS SP <2>	WCS <111>	Spare Bit	

NANO-INSTRUCTION

The microcontroller sends a nano-instruction and its related information to all the matrix boards and the io boards in the system during each clock cycle. The nano-instruction is made up of 3-bit opcode and a 16-bit (plus a parity bit) instruction. The 16-bit instruction is in turn made up of different fields with opcode dependent definitions. Please refer to CM CHIP (Beta) SPECIFICATION for detailed description of each field and its function.

OP NAME	0 NOP	1 LOADA	2 LOADB	3 STORE	4 READ	5 LOADI	6 ROUTE	7 RUG
I0	DC	FLAGR0	COND0	FLAGW0	Z	DATA0	CYCLE0	R/W
I1	DC	FLAGR1	COND1	FLAGW1	Z	DATA1	CYCLE1	A/C
I2	DC	FLAGR2	COND2	FLAGW2	Z	DATA2	CYCLE2	NEWS
I3	DC	BSEL	INV	MB0	Z	DATA3	CYCLE3	MB0
I4	DC	ALUC0	ALUS0	MB0	Z	DATA4	CYCLE4	REG0
I5	DC	ALUC1	ALUS1	MB0	Z	DATA5	CYCLE5	REG1
I6	DC	ALUC2	ALUS2	MB0	Z	DATA6	CHECK0	REG2
I7	DC	ALUC3	ALUS3	EDGE	Z	DATA7	CHECK1	REG3
I8	DC	ALUC4	ALUS4	CUBE0	Z	DATA8	XOR1	REG4
I9	DC	ALUC5	ALUS5	CUBE1	Z	DATA9	XOR2	MB0
I10	DC	ALUC6	ALUS6	CUBE2	Z	DATA10	XOR3	MB0
I11	DC	ALUC7	ALUS7	CUBE3	Z	DATA11	XOR4	MB0
I12	DC	MB0	MB0	MB0	Z	DATA12	SNARF0	MB0
I13	DC	MB0	MB0	MB0	Z	DATA13	SNARF1	MB0
I14	DC	MB0	MB0	MB0	Z	DATA14	SNARF2	MB0
I15	DC	PMODE	PMODE	PMODE	Z	DATA15	ODD	PMODE
I16	DC	PARITY	PARITY	PARITY	Z	PARITY	PARITY	PARITY

DC means Don't-care. MB0 means must be zero. Z means high impedance.

The 16-bit instruction is commonly referred to as shown below.

i) I<3:0> FLAG field

ii) I<11:4> ALU field

iii) I<15:12> PMODE field

The other information associated with each nano-instruction includes addresses and control signals.

1) CHIP SEL

The CHIP SEL field is 12-bit field inside the microcontroller. These bits are the most significant 12 bits of the 32 bit address used for direct memory read/write access. This field is split into two sub-fields.

1.1) CADDR <4:0>

This field, CHIP SEL <4:0>, is used for selecting one of thirty two CM chips on the matrix board.

1.2) BOARD SEL

Each matrix board receives a board select line. The microcontroller drives 1 of 16 board select lines by decoding the CHIP SEL <11:5> field.

2) MADDR <19:0>

This field is used as the memory address to access the memory associated with each processor. Although, the first beta machine will have 64Kbit memory per processor, the 20-bit field is provided for future upgrading to 1Mbits. For direct memory read/write access, these bits make up the least significant 20 bits of the 32-bit address.

3) WE HI, WE LO

These two bits control the WE (write enable) signal of the high and low bytes of the memory.

4) SENDREVEN, SENDRODD

These two signals control the cube output drivers of the even and odd CM chips.

5) LATCHREVN, LATCHRODD

These two signals control the cube receiving latches of the even and odd CM chips.

6) LATCH LED

This signal controls the latching of the LED on the matrix board.

7) NEW PAGE

This signal indicates that new page is detected.

8) DIM <11:8> ENA

These four signals are decoded from the Nexus Cube Enable signals. These signals indicate whether the dimensions 8, 9, 10 and 11 are enabled.

9) BID EVEN/ODD

These signals are used for board identification. Each matrix board receives either BID EVEN or BID ODD signal.

10) REG SEL

This bit, along with CADDR <4:0>, is used for reading registers on the matrix board. The bit is set to read registers and when the bit is not set, the CADDR has its normal meaning.

11) LD STAT

This bit, when active, indicates that the current data in the FP field corresponds to the FP STATIC field and this data must be latched in the matrix board. This bit is asserted when the UW LIT VERT = 4.

12) STALL

This bit, when active, indicates that the current data in the FP Dynamic Field is not valid data and the current Weitek cycle must be killed. This bit is asserted whenever the UW LIT VERT is not equal to 5.

13) REFRESH

This signal generates the refresh for the dynamic RAMs on the matrix board.

14) SPRINT OP

This is the three bit SPRINT chip opcode field. Please refer to the SPRINT user's manual for details.

15) FP INSTRUCTION

This is a 24-bit field. This field is used for sending instructions to the Weitek MAC. The Weitek instruction is divided into two parts, dynamic and static. The static field is 20 bits wide and it is latched on the matrix board. The dynamic field is 24 bits wide it can be changed on every cycle. The static and dynamic fields, together, make the entire Weitek instruction.

16) IO READ A, IO READ B

These signals enable the IO latches on the matrix board to transfer data from matrix board to the IO board. (A & B stands for the left and right side of the backplane respectively.)

17) IO LOAD A, IO LOAD B

These signals latch the data on the matrix board, sent to it by the Io board.

STATUS REGISTERS

The registers on the Nexus and micro controller are defined by bits RSEL<3:0>.

RSEL<3:0>	Name	location	operation
0	NOP		NOP
1	write IFIFO	UC	write to Input FIFO
2	write SR0	UC	write to Stat Reg 0
3	-	-	-
4	read OFIFO	UC	read from Output FIFO
5	read SR0	UC	read from Stat Reg 0
6	read SR1	UC	read from Stat Reg 1
7	-	-	-
8	read NSR	Nexus	read from Stat Reg
9	write NSR_L	Nexus	write to Stat Reg (low)
10	-	-	-
11	write NSR_H	Nexus	write to Stat Reg (high)
12	read WCSA	UC	read WCS Address
13	write WCSA	UC	write WCS Address
14	read WCSD	UC	read WCS Data
15	write WCSD	UC	write WCS Data

The registers on the micro controller are defined below. (Only the interesting ones for now. When time permits, more will be written.)

Status Reg 0 (RSEL = 2 for write, RSEL = 5 for read)

Bit 0 - RUN
The RUN bit allows the host the ability to stop/start the UC. When deasserted, the UC is stopped. When asserted, the UC is running. This bit is deasserted by power on initialization, a CM generated trap, or when the host asserts the HM_RST bit (bit 1 of this register). The RUN bit must be deasserted in order to read or write the contents of the WCS. This bit is read/write.

Bit 1 - HM_RST
When asserted, this bit allows the host the ability to reset the UC. both the IFIFO and the OFIFO are reset by the assertion of this bit. In addition, the assertion of this bit clears the RUN bit (bit 0 of this register). This bit is write only. When read, it should always be deasserted. If it is not, then it is an indication that the HM_RESET line on the UC is stuck high.

Bit 2 - HM_RST_ERR
When asserted, this bit allows the host to reset all errors detected in the UC. It will re-enable the logic that saves the micro address and the micro instruction as code is executed (i.e. the logic that saves the state of the UC on a trap). This bit is write only. When read, it should always be deasserted. If it is not, then it is an indication that the HM_RST_ERR line on the UC is stuck high.

Bit 3 - DIAG_MODE
The assertion of this bit puts the UC in a diagnostic mode, forcing the OP field and the control fields sent the the matrix and IO boards to be a NOP. This allows the UC diagnostics to be executed without disturbing the CM. This bit is read/write.

Bit 4 - BREAK
When asserted, this bit informs the host that the UC has encountered a breakpoint. This bit is read only. A host write to this bit has no effect.

Bit 5 - CONT
The host asserts this bit in order to continue from a breakpoint in the micro code. This condition is detected when the BREAK bit (bit 4 of this register) is set. This bit is write only. When read, this bit should always be deasserted. If it is not, then it is an indication that the HM_CONT line on the UC is stuck high.

Bit 6 - MASK_UC_ERR
When asserted, the host can mask all error conditions that are

generated on the UC. These are detailed in bits <17:5> of Status Reg 1. This bit is read/write.

Bit 7 - MASK_CM_ERR
When asserted, the host can mask all error conditions that are generated on the CM. These are detailed in bits <2:1> of Status Reg 1. This bit is read/write.

Bit 8 - MASK_IO_ERR
When asserted, the host can mask all error conditions that are generated on the IO boards. These are detailed in bits <4:3> of Status Reg 1. This bit is read/write.

Bit 9 - WCS_REG_CNTL
When asserted, this bit allows the host to read the saved data from the WCSA (RSEL = 12) and the WCSD (RSEL = 14) registers. When deasserted, the WCSA will display the address of the control store and the WCSD will display the addressed contents of the control store (if RUN = 0). This bit is read/write.

Bits <12:10> - HOST_TRAP
This field is used to generate a trap vector to the UC. It provides an alternate means (other than through the IFIFO) for the host to communicate with the UC. Trap routines are typically used for error handling or diagnostic functions. These bits are write only. When read, these bits are always deasserted. This field is encoded as follows:

bits<12:10>	
0	NOP
1	NOP
2	NOP
3	NOP
4	Host trap 4, vector 104 (hex)
5	Host trap 5, vector 105 (hex)
6	Host trap 6, vector 106 (hex)
7	Host trap 7, vector 107 (hex)

Bits <14:13> - MEM_SIZE
These two bits are used to define the size of memory chips on the matrix boards to the micro controller. This information is used to support the page mode access of these dynamic memories. The two bits are read/write by the host and may be written by the micro controller. A power up reset, or a host reset (caused by the host asserting bit 1 of this register) will cause these two bits to be deasserted. The bits are encoded as follows:

Bits <14:13>	memory chip size
0	64 K
1	256 K
2	1 M
3	reserved

Bit 15 - NEXUS_IO_RDY
This bit reflects the state of the composite IO Ready line received from the Nexus. This bit is read only.

Bit 16 - UC_IFIR
This bit, when asserted, defines that the micro controllers input FIFO is ready to accept data from the host. This bit is read only.

Bit 17 - UC_IFHF
This bit, when asserted, defines that the micro controllers input FIFO is at least half full. This bit is read only.

Bit 18 - UC_IFOR
This bit, when asserted, defines that the input FIFO has data in it that is available to the micro controller. This bit is read only.

Bit 19 - UC_OFIR
This bit, when asserted, defines that the output FIFO is ready to accept data written to it by the micro controller. This bit is read only.

Bit 20 - UC_OFHF
This bit, when asserted, defines that the output FIFO is at least half full. If deasserted, then the output FIFO is less than half full. This bit is read only.

Bit 21 - UC_OFOR
This bit, when asserted, defines that the output FIFO has data available to be read by the host. This bit is read only.

Bit 22- NEXUS_IF_EMPTY
This is the composite input FIFO empty signal from all micros. This bit is read only.

Bit 23 - NEXUS_OF_EMPTY
This is the composite output FIFO empty signal from all micros. This bit is read only.

Bit 24 - NEXUS_OFHF
This is the composite output FIFO half full signal from all micros. This bit is read only.

Bit 25 - SAVED_GLOBAL
This bit reflects the state of the Saved_Global flip-flop in the UC. This bit is read only.

Bit 26 - NEXUS_RECV_GLOBAL
This bit reflects the state of the composite Global from all micros. This bit is read only.

Bit <28:27> - NEXUS_DIM_ENB<1:0>
These bits reflect the state of the dimension enable lines received from the Nexus. These bits are read only.

Bits <30:29> - UC_ID<1:0>
These bits reflect the state of the UC ID wires received from the Nexus. These bits are read only.

Bit 31 - NEXUS_IFHF
This is the composite input FIFO half full signal from all micros. This bit is read only.

Status Reg 1 (RSEL = 6 for read)

This is a read only status register. It reflects the state of all bits in the UC that cause traps to occur.

Bit 0 - UC_ERR
This bit is the composite of all errors detected on the UC. That is, it is the logical OR of bits <17:5> in this register.

Bit 1 - CM_ERR_0
This bit is the global error that comes from all matrix boards in the upper backplane of the CM quarter.

Bit 2 - CM_ERR_1
This bit is the global error that comes from all matrix boards in the lower backplane of the CM quarter.

Bit 3 - IO_ERR_0
This bit is the global error that comes from all IO boards in the upper backplane of the CM quarter.

Bit 4 - IO_ERR_1
This bit is the global error that comes from all IO boards in the lower backplane of the CM quarter.

Bit 5 - UC_UW_PARITY_ERR
This bit is asserted if a parity error is detected in the micro instruction that is to be executed.

Bit 6 - IFIFO_PARITY_ERR
This bit is asserted if a parity error is detected at the output of the input FIFO.

Bit 7 - MADDR_LO_BND_CHK_ERR
This bit is asserted if a bounds check error was detected when comparing the MADDR lines with the lower bound check register.

Bit 8 - MADDR_HI_BND_CHK_ERR
This bit is asserted if a bounds check error was detected when comparing the MADDR lines with the upper bound check register.

Bit 9 - RDATA_PARITY_ERR_0
If asserted, this bit indicates that a parity error was detected on bits <7:0> of RDATA.

Bit 10 - RDATA_PARITY_ERR_1
If asserted, this bit indicates that a parity error was detected on bits <15:8> of RDATA.

Bit 11 - SRAM_OVFL_ERR
If asserted, the scratch ram address counter was loaded with a value, or incremented to a value greater than the amount of scratch ram available.

Bit 12 - SRAM_UFLW_ERR
If asserted, the scratch ram address counter was loaded with a value, or decremented to a value less than zero.

Bit 13 - SEQ_STK_FULL_ERR
If asserted, the micro sequencer subroutine stack has been filled. The stack is 33 locations deep. When a a push to the 33d location will cause this error to occur. Therefore, only 32 locations be used.

Bit 14 - SRAM_PARITY_ERR
When asserted, a parity error occurred while reading the scratch ram. (note: this feature may not be implemented if there aren't enough nano-acres on the UC board.)

Bit 15 - THERMAL_WARNING_0
If set, this bit indicates that a thermal warning was generated on one of the boards in the upper backplane of the CM quarter.

Bit 16 - THERMAL_WARNING_1
If set, this bit indicates that a thermal warning was generated on one of the boards in the lower backplane of the CM quarter.

Bit 17 - UC_TRAP
If asserted, this bit confirms that the trap was caused by the microcode. It is intended that this will be used by the micro code to flag software detected errors (e.g. divide by zero).

Bits <30:18> - reserved

Bit 31 - NEXUS_COMP_ERR
This is the composite error signal from all micros.

TRAPS

The trap mechanism of the micro controller is intended primarily for error handling. They can be generated from the hardware detection of an error, from the micro code, or from the host. A trap causes the micro code to take an unconditional jump. The current micro program counter is NOT preserved. This means that there are NO returns associated with traps. The trap will cause an unconditional jump to the trap vectors as shown in the following table, starting with the highest priority.

hex addr (vector)	condition	RUN bit	TRAP_ENBL flag	error state
100	Nexus Composite Error	cleared	cleared	saved
101	reserved			
102	reserved			
103	ucode trap	cleared	cleared	saved
104	Host Trap 4	unchanged	unchanged	not saved
105	Host Trap 5	unchanged	unchanged	not saved
106	Host Trap 6	unchanged	unchanged	not saved
107	Host Trap 7	unchanged	unchanged	not saved

Trap vectors 100 and 103 clear the RUN bit in the host status reg 0. This will cause the UC to stop executing micro code. The host can then examine the error state of the UC and decide how to proceed. The host will set the RUN bit to execute the trap routine. This may simply be a "jump to self" piece of code. The host can then direct the UC to the appropriate location with Host Traps 4, 5, 6, or 7. These are generated by the host writing to bits <12:10> of status reg 0.

There are three major classes of errors.

1. CM errors
2. IO errors
3. UC errors

The host has the ability to mask any of these classes of errors using bits <8:6> of status reg 0. In addition, there is a single TRAP_ENBL flag that the micro controller can set or clear. When trap 100 or 103 occurs, this flag is automatically cleared, inhibiting further traps. The micro code should enable this flag on exiting from the trap routine. Alternately, the host will enable the flag when HM_RST_ERR is generated by asserting bit<2> in status reg 0.

Various error conditions are saved for the traps that vector to addresses 100 and 103. The host can read these conditions from status reg 1. In addition, the host can read the micro address and the micro instruction of the instruction where the trap occurred. This is done by asserting the WCS_REG_CNTL bit in status reg 0. Then, the micro address of the instruction that caused the trap can be read from the WCSA register. The micro instruction can be read by the host when four successive reads of the WCSD register are performed. This will yield bits <31:0>, <63:32>, <95:64>, <111:96> in each read respectively. The error state saving logic will be re-enabled when the host asserts the HST_RST_ERR bit of status reg 0.

LOADING WCS

The mechanism for loading micro code into the writable control store (WCS) is somewhat different in the beta UC (compared to the alpha UC). The host should first insure that the RUN bit in status reg 0 is deasserted. This will enable the WCS to be loaded. The host should write the address of the first instruction (i) to be loaded to the WCSA register (RSEL = 13). This will address the WCS. The host should next write the instruction that resides at this address to the WCSD register (RSEL = 15). However, because an instruction contains more bits than does the register, the host must write the data in four pieces. The first write to WCSD will load bits <31:0>, the second write will load bits <63:32>, the third write will load bits <95:64> and the fourth write will load bits <111: 96>. In order to load the next instruction (i+1) into the WCS the host must again write the address (i+1) to the WCSA register before doing four writes to the WCSD register.

The host can do a read-verify of the data in the WCS by performing a similar set of operations. The WCSA is loaded with the address of the location to be verified. Four successive reads are then performed from the WCSD to get bits <31:0>, <63:32>, <95:64>, <111:96> of the instruction. It should be noted that during such a read operation the WCS_REG_CNTL bit of status reg 0 should be deasserted.

IO CONTROL

TBD.

RESTRICTIONS and APPLICATION NOTES

1) Either a read or a write can be performed on the Scratch RAM during a cycle, but not both.

2) Sourcing of the performance monitor data (least significant 32 bits or most significant 8 bits) on the D-bus and write operation in to the scratch RAM (data from Y-bus) can not be done in the same cycle.

   Similarly, sourcing of scratch RAM data (read operation) on the D-bus and loading of the performance monitor (data from Y-bus) can not be done in the same cycle.

3) The RAM shifts and Q shifts for the micro-controller ALU are simple shifts and not rotates. (Previously, (till 2/25/87) these were implemented as rotates.)

Appendix A: Hardware Wishlist

As a base to start from, the following list of hardware requirements will be considered in designing the beta version of the micro controller.

First, those items that must be done.

1. All signals between the UC and the Matrix board must be parity protected.

2. Systems configurations of the beta machine must be able to support two UC boards in each backplane. Each UC will control the matrix boards in half of the backplane. This requirement will enable us to ship smaller machines.

3. Because of added signals that must be supported, the Gbus interface to the UC will consist of 3, 50 pin flat ribbon cables.

4. All address flip logic goes away.

5. All zipper stuff goes away.

6. When the host does a reset of the CM, this must not affect the contents of the Drams on the Matrix boards.

7. Information contained in the status registers (read by the host) must be increased. This will provide more information for testing the UC. For example, individual board FIFO ready bits should be reported rather than just the composite ready bit received from the NEXUS. There should also be two error mask bits. One will mask UC errors, the other will mask errors received from the Matrix boards.

8. Error trap handling needs to be analyzed and cleaned up.

9. An ID jumper (or switch) needs to be placed on the UC. This will replace the ID jumper that exists on the backplane in the alpha machine.

The following items are on the wish list with the highest priority item listed first.

1. The design should progress as quickly as possible with the schedule considered as the top priority.

2. The UC should be designed to run at 8 Mhz. However, the design goal should be to make the UC run at 10 Mhz. This will provide us the future speed that will most surely be needed.

3. The UC must be at least as debuggable as the current UC. This also implies that it be stand alone debuggable.

4. It is desired that the UC contain a dozen (or so) spare IC locations.

5. It is desired that all pals have some spare inputs and outputs. This will allow us room should design changes be required.

6. The instruction, maddr, and most control lines to the matrix board should not be allowed to change if the instruction is a NOP (op = 0). This will keep the noise to a minimum throughout the system.

7. Provisions should be made to hook up some sort of a performance monitor. How this is implemented and where it will be located is TBD. However, provisions must be made to allow control of the performance monitor from the host via the Gbus interface.

8. It might be useful to include a "mark" bit in the micro instruction. This will provide a hardware trigger mechanism for testing and perhaps for use with the performance monitor.

Appendix B: Software Wishlist

This note summarizes the system software requirements for the Beta microcontroller. It is based upon inputs received from a number of people and is still subject to change if significant items were omitted. Beside the obvious goal of supporting the Beta chip, these requirements reflect the following goals.

1. The Beta microcontroller should allow all microcode to run on the microcontroller. Currently, due to a number of hardware limitations some code which should run on the micro runs on the host. This makes multiple hosting and multiple language interfaces more difficult to achieve and maintain.
2. The Beta microcontroller must support a number of new, or extended system features. The system will support "timesharing" of CM sections. An extended flexible VP scheme which allows for the use of many VP ratios with different memory sizes for VPs will be implemented. The I/O system will require control and monitoring from the microcontroller. The SPRINT chip subsystem will require additional control.
3. The Beta Microcontroller should efficiently support highly parameterized microcode. Currently, there is a separate version of the microcode for each VP ratio. Since we want to have a more flexible VP scheme, and time sharing and I/O, the number of variations of microcode objects would become unmanageable. Thus, the microcontroller must have adequate support for storing an accessing parameters.

The assumptions supporting the Beta microcontroller requirements are:

1. As little software and microcode as possible should be required to change in order to run on the Beta microcontroller. That does not mean that future optimizations which may require significant changes should be precluded.

2. The beta microcontroller must support the beta chip. Thus 8MHZ operation along with at least 64kbit addressing with ECC on memory is required. The use of DRAM may require some software optimizations in order to get performance close to the 8MHZ clock rate. Also, the microcontroller should be able to run at selectable lower speeds.

3. The BETA microcontroller will not be used on Alpha machines.

BASELINE REQUIREMENTS

1. More control over SendR and LatchR. The test code for :Which-Dims-Receive needs to be able to generate SendREven and LatchREven together, and SendROdd and LatchROdd together. Also need a way to have no SendR or LatchR. The proposed fix is to add a ROUTER-CNTL-MSB bit and make:
(DEFUC-FIELD ROUTER-CNTL ((+ 12 32) 2) 0
" This field controls the generation of the SENDREVEN, SENDRODD, LATCHREVN, and LATCHRODD signals.

ROUTER CNTL	ROUTER CNTL MSB	SENDREVEN	SENDRODD	LATCHREVN	LATCHRODD
0	0	1	0	0	0
1	0	1	0	0	1
2	0	0	1	1	0
3	0	0	1	0	0
		Instruction output feedback to read data path			
0	1	0	0	0	0
1	1	1	0	1	0
2	1	0	1	0	1
3	1	Reserved"			

)

2. Support for 8MHZ operation and DRAMS - At 8MHZ the use of DRAMS requires the use of a repeated CAS cycles in order not cause the insertion of processor wait states. The microcontroller must recognize that consecutive addresses reside in the same or different RAS pages. For those consecutive addresses that reside in the same RAS page, the microcontroller should use consecutive CAS cycles without any intervening RAS cycles. For other cases, the microcontroller will have to do RAS followed by CAS cycles while inserting wait states as needed. This should be transparent to both microcode and higher order software.

Future compilers will attempt to optimize memory layout so that operands which are most frequently used in an operations are in the same RAS page (256 bits for 256kbit DRAMS, 512 for 1Mbit, and 1024 for 4Mbit ones).

3. Read and Write flippers disabled and removed. The only flipping in the Beta chip will be on Store instructions and will be controlled by a flip rug register.

4. IFLIP-SEL removed and replaced by a 4 bit microword PMODE field and a Mux that is controlled by ARG-INSTR:
    0 - uw-PMODE
    1 - (logxor (uw-PMODE IFA))

5 . AFLIP-SEL removed. The AFLIP-SEL field in the micro-instruction can be scavenged.

REQUIRED FEATURES

1. Memory - The Beta microcontroller should have at least 64k words of memory. Currently, almost every word of the 16k microstore is used. Some PARIS instructions were dropped; standard math functions like sin, exp, log, etc should be microcoded; floating point should be IEEE compatible; the beta chip has new grid and
router functions; and i/o will need some space. In addition, a separate section of microstore is needed for user defined instructions.

2. The microcontroller will have scratch pad memory that can be addressed either with the stack register or the scratch memory address register. There should be at least 4k words of scratch pad memory. This memory's output would be used as ALU input. The
most frequent use of this memory will be for storing tables that will be used to control the VP sizes and displacements and per user start and end addresses for time sharing. Some other
potential uses are storing tables of constants, creating a microcode subroutine stack, gathering performance data, and probably storage of state information about I/O operations.

3. The ALU should have access to at least 32 registers (64 is highly desirable).

4. A "stack register" - This register would support post-increment and pre-decrement, and setting from the ALU output or the microword. This register which could make subroutine calling in microcode feasible would address the scratch memory.

5. A scratch pad memory access register which can be set either from the microword or the output of the ALU. It will be used for accessing the scratch pad memory.

6. In order to support the larger RAMs (beyond 64k per processor), the data path should be at least 20 bits wide. With 20 bits, the proposed 4Mbit DRAMs can be supported (1 Mbit per processor).

7. In order to support multiple users of a Connection Machine section in a safe and sane manner, one should be able to compare the MADDR with a pair of upper and lower memory address limits. If any MADDR is not within the limits of these 2 addresses, then a memory limit violation error is signaled. These limit register should be settable from the output of the ALU. The lower 8 bits of the MADDR need not be compared since per user memory will be allocated in 256 bit pieces starting on a multiple of 256. Finally, the limit comparison should be able to be enabled or disabled on a per microinstruction basis in order to allow the system firmware to use physical memory without restrictions. It is highly desirable that the microword have a field that specifies that the lower limit register be used as a direct input to the ALU.

8. Need a way to generate bad instruction parity, in order to test that part of the chip error system.

9. In order to eliminate parts of the Alpha micro such as the Shift register, the modcounter, and the Shmud, we need test and branch conditions on the most and least significant bits of the Q register and a flip flop to detect odd and even cycles.

10. In order to check the integrity of the data being sent through the OFIFO, there should be a way generate parity on the data going into the

OFIFO. The current way of sending the data and the data shifted is not appropriate.

DESIRED FEATURES (in decreasing order of need)

1. A readback path from the CM memory to the microcontroller would be an effective way of increasing microcontroller scratch pad memory. As an alternative to the current use of the RDATA path, data would be fetched from the same relative chip and memory address in each microcontroller's section. For operations like reading a processors memory, the microcontrollers would then be responsible for passing back only the correct data to the OFIFO. For the case where the CM memory is being used to alter the execution sequence in the micros, the software and firmware will guarantee that the data is identical in each memory location that may potentially be read back. The data read will be used to identically alter the subsequent actions of the microcontrollers.

2. A way of using the Q register and the Global line to create a single value in all micros from micros having different values. Consider the case where the 2 micros get status from 1 I/O controller in each of their sections. In order to determine the complete status of the I/O subsystem the micros must combine the status of the two I/O controllers. The proposal would use the Q register to store the status values. Then, the Q reg would be shifted such that the LSB of the Q reg is combined on the global line going to the nexus. This combined value would then be shifted into the MSB of the Q reg. After the appropriate number of shifts, the micros would have the same value.

3. A basic check on the synchronism of the microcontrollers should be done. An example would be in the NEXUS to compare the parity of the certain portions of the ALU output and the microPC. With the introduction of more asynchrony (from I/O devices), the possibility of multiple micros getting out of sync increases greatly.

4. A cycle count register (at least 40 bits long) that can be latched in one cycle and read from the latch into 2 scratch pad memory locations or two ALU registers in 2 cycles would be useful for performance monitoring. This register should be able to be cleared in one cycle, and be able to be set to either increment or decrement on every microcycle thereafter. It is also desirable that the incrementing or decrementing be controllable on a per microinstruction basis. Finally, although of lesser importance, it is desirable to have a testable, clearable flag which is set and held at logical 1 when the register counts to 0. This register and flag could then be used to time long asynchronous events like I/O in order to detect errors.

5. A read only status bit indicating whether the microcontroller is executing from PROM or RAM.

6. A second set of limit registers would help in a number of areas such as VP bounds checking.

SPECULATIVE REQUIREMENTS

1. For the I/O system, the microcontroller should be able to communicate directly to the I/O controllers. At a minimum, each I/O controller (there could be up to 1 per backplane (a total of 8)) should be able to set a flag which the microcontroller(s) could read at the end of macroinstructions; this could be used like an interrupt in order to redirect the microcontrollers' next sequence of instructions. Since from 1 to 4 microcontrollers may be interested in any number of the I/O controllers, these flags must be brought to each micro. It would be beneficial if they selectable subsets of these flags could be ANDed and used in simple tests. In addition, a path is needed that allows the microcontroller(s) and the I/O device controllers to exchange information. For example, the I/O controller may need to tell the microcontroller(s) that an operation failed so that no further data movements be attempted. Similarly, microcontroller(s) may need to start or stop some operations due to any number of circumstances.

2. While the SPRINT chip is not fully defined, it is likely to have some impact on the the microcontroller. One likely effect of indirect addressing is that since different processors may each be accessing

different memory addresses all indirect references will need to do complete RAS/CAS cycles even though the microcontroller sees 2 consecutive addresses as being in the same RAS page.
Furthermore, the sprint chip will create the need to have new CM opcodes. Space should be reserved for them.
Finally, it appears that 4 5 bit counters are needed to control the FP chip register file.

REMOVABLES (see conditions above)

1. The Flippers can be removed

2. The Shmud can be removed.

3. The Modcounter can be removed

4. IFLIP-SEL and AFLIP-SEL can be scavenged.

5. The IFLIP-SEL mux can be removed

6. The Shift register can be removed

7. GDIR SORW can be removed.

Appendix C: FP Wishlist

Since we have limitation on number of wires on the backplane and the Weitek chip needs a lot of bits to control it, we suggest that the control word of the weitek instruction get split into 2 groups of fields:

DYNAMIC and STATIC (were dynamic can change every cycle, and static can not).

An 18-bit latch on the matrix board latches the static field. Most operations don't change this often. This latching is controlled by a bit in the microword.

The static and dynamic fields are overlaid in the microword. Hence one cannot specify both in the same cycle.

The fields are as follows:

Static Fields (total 18 bits)

Processor 3164:

|Func(4)|Aain Sel(2)|Abin Sel(3)|Alu Dest(2)|Main(2)|Mbin(2)|Cons(2)|PS(1)|

Processor 3132:

|Func(3)|NC(3)| Abin Sel(3)|Alu Dest(2)|NC(3) |Mbin(1)|Encn(1)|NC(2)|

Dynamic Fields (total 24 bits)

Processor 3164

|A-addr(5)|B-addr(5)|C-addr(5)|D-addr(5)|IO-cntl(3)| CWEN(1)|

Processor 3132:

|A-addr(5)|B-addr(5)|C-addr(5)|D-addr(5)|IO-cntl(2)|NC(1)|CWEN(1)|

What the sprint and the weitek need for control:

1) These 24 bits (worst case of static and dynamic lengths) from weitek
2) 1 bit to control the latch
3) 1 bit to control whether indirect addressing will be used
4) 1 bit to control whether RAS/CAS or just CAS will be used in indirect addressing
3) 3 op pins for sprint chip (instruction lines are shared with CM chips)

This assumes a 18 bit latch on a matrix card.

Appendix D: Design Review (11/14/86) Minutes

****
The only action items need to be resolved at this time (11/21/86)
are 23 and 26. Both of which depend upon the board real estate availability.
****

The minutes of the beta microcontroller design review follows. Action items are noted with the names of the people responsible for follow-up. The first name shown is the owner. The other names are probable persons involved in the decision. Given that this was done after the meeting, if you don't feel as though you are responsible for the follow-up, find someone else to take your place and let us know.

>>>> In the absence of an answer, we'll take our best guess. <<<<

1. Does the literal field need to be 16, 18, 20, or 32 bits. It was decided that a 20 bit and a 32 bit literal field would be needed. The IO_Cntl field and the Router_Cntl field will exchange places in the micro word definition. The UW_Lit_Vert field would be increased to 3 bits to support the added literal field. i.e.:

Lit Vert	31 24	23 21	20 18	17 15	0
0	32 bit literal field				
1	ALU field	Router Cntl	20 bit literal field		
2	ALU field	Router Cntl	IO Cntl	NA field	
2	ALU field	Router Cntl	IO Cntl	*	iteration count
3	ALU field	Router Cntl	IO Cntl	FP Static field	
4	ALU field	FP Dynamic field			

If possible another encoding will be provided to support a 16 bit literal field. i.e.:

ALU field	Router Cntl	IO Cntl	16 bit literal field

2. Selection of UW_Lit_Vert = 3 will cause the FP static field to be latched in the matrix card. Implicit in this is the case that if this field is not encoded to 3, the signal is not latched. This will be noted in the spec. Similarly, all control fields will be NOPed when not used. The NOP condition will be specified and, where possible, will be encoded as zero.

3. The spec will change all instances of the term "Router control" to "Cube control".

4. A desire was expressed to have both IO_Cntl and a 16 bit literal available in the same instruction. This issue needs further study to better understand the need. (see item 1 above)
***Action item: Jim L. & Marshall

5. A comment was made about the need to select multiple IO boards in the same micro instruction. This feature will be available.

6. The CM_direct_R/W_access encoding will be removed from the IO_Cntl<2:0> field. This will free up an encoding in this field. The function will be moved to UW_ENC_FLD_B.

7. Should OF_Full be replaced by OF_Half_Full? The subject was debated and the consensus was that it should. Hence forth OF_Half_full will be used in place of OFIR (not (OF_Full)).

8. The shift out bits of the 49C402A ram register will be available for a branch condition. Specifically, RAM_0 will be connected to RAM_31 to implement an end-around shifter (rotator). The state of this line will be saved at the end of each instruction for use as a branch condition in the next instruction.

Beta Chip Checkout

12/2/86 9 chips recieved 10 AM 4 MHz

- SN #1 works on many tests except a few processor-wise reads.

reassembled and everything works

except:

Backward-tester-3

The error is processor 4/56 got 0 rather than 123

- trying SN #2, fails same
All other tests OK

* to track down
why does determine-parity-mode time out in Rel-2 but passes in diagnostics

- SN #2 runs at 5 MHz as well, ... fails same way.
runs at 2 MHz as well --- fails same way.
NO: ECC, Error system
- SN #3 fails same way.

Note: Changed CM:INIT -I Added Route with :snarf :match]

Fixed problem in Backward-tester-3
12/2/86 16:30

Micro REV-4 mods.

Removed	E20-5	E11-7
Removed	E20-5	E21-15
Added	E20-5	E21-18
Added	E21-19	E11-7

This connects Masked err to the RST Pal instead of comp err. so that the micro can still play when a cm error occurs. (Comp err would disable the UW CLK signals)

"PARITY-ECC-TESTER.LISP" 12/3/86 11:50
Changed several "24."s to 22
Inserted bit portion of CM-REC-ERR

Reproduce Instruction parity error?

(Buffer-Global-Tester) Parity/ECC mode problem.

Same problem: (Hatch box-tester)

Possible Chip Problem: Route cycle in ECC mode gets parity error. Works OK in parity mode.

12-3-86

Things to try:

✓ Memory tests in ECC

Tests with broken ECC

A) Remove wire

B) Short output ?

Microcontroller P.C.

(TEST-CM-ERROR) [Force Parity Error]

Add mode check at end of loop. [Parity error ?]

✓ Change Core Read & Write to not use CM-TEMP-BIT-0

✓ Generate-Delivery-Code;

No — Use RPCT for clear loop with PMODE=1

Got around — At end, re-init mod cntr & clear buffer again

✓ Test LED. Hack on Mating board

Check power-up sequence with entire system.

12-4-86

(Set-Message-P): Fixed - conflicting use of CM-TEMP-BIT-0

(Set-Match-Box): Fixed - same type of problem

(Buffer-Global-tester) — FIXED
(Match-Box-tester) — OKAY

Problems: (NEWS-ID-TESTER)
(RUG-NEWS-TESTER-PARALLEL)
(FUNCTION-BOX-TESTER)

→ Do a (Clear-Memory) after both (SET-PARITY-MODE) and (SET-ECC-MODE) (FUNCTION-BOX-TESTER) — FIXED

Initialize test pattern *SAFELY*.
(RUG-NEWS-TESTER-PARALLEL) - FIXED
(NEWS-ID-TESTER) - FIXED
(" -20- ") - OKAY
(EJECTOR-TESTER-PARALLEL) - OK
- (MSG-PARITY-TESTER-PARALLEL)

NOTE

(INITIALIZE-RVG-ALWAYS) Bombs in ECC mode [Mem parity error]
Forces parity mode - Use with
EXTREME CAUTION

Up to Rel-2 is Okay
YEAH!!

12/4/86 10:30

Backward-Route-225 still works.

Generate debug code is broken but don't quite know how to fix. [see source]. (missing Q reg?)

When in doubt, see why you're using CP-TERM-BIT-B.

12/5/86

(TEST-WRITE-READ-WEXO-WE##)
doesn't work in ECC mode -
can't work in ECC - changed
to force parity mode during test.

12/15/86

Speed checkout
VCC ⇒ 7.25 MHz (DO-Repeat-...)

(~~system~~-test :vcc-tests nil)

5.5 ✓ 5.70 ×
5.65 ?
6.0

1/6/87 Full Board Test

- Rug-Tester Select test failed one time on Heart on clip 0/0
  - could not reproduce

- News-2D-Tester doesn't work for 1 board
  - Temporarily removed

1/7/87 Beta Resources

nex 3-16	(TB4)
M3-127	(TB4)
M3-W548	(TB4)
UC 4-1	(TB4)
nex 3-1	(mungo's TB) (TB2)
EX-10	TB4
EX-73	TB4

1/9/87

2 boards

Router - Tester - Fast

A — #0444 proc got #0444 #0464
picked up #020
chip 0/18 dir 703

B — #0501 picked up #020
chip 0/20 dir 726

C — #0643 picked up #020
chip 0/26 dir 821

D — #01327 picked up #010
chip 1/13 dir 827

E — #0253 picked up #020
chip 0/10 dir 887

Router - Tester - Scrambled - Fast

— A above
— #0711 picked up #020 0/28
— #0617 " " 0/24
— #0613 " " 0/24
— #0656 " " 0/26

Works at 2MHz

4-5/87 D-Sat. Came in to find
Bnt was doing fairly well. Board
A10 faulted DIM 7 parity error repeatably.
Will swap A10 and A11 to see if problem
tracks bd.

Nexus still loses on bit 22 GBUSD while
running GBUSC.

Swapped in new nexus to see if new
one would fail. It fails, but on
GBUSD D<18> while running GBUSC
could not loop on holding bit 18 low.
Swapped orig nexus back in and
started (S/... test) on Bnt A
and test ... on Bnt B ...
... ...
... or Rack.

Mungo is Ø-C ← UC11

Rock is 1-D ← UC12 ← Fails bit 22 GBUS D

Mungo is 1-C

Rock is Ø-D ← Fails bit 22 GBUS D

~~ROCK IS~~ [MUNGO is 1-D ← Fails bit 22/18
ROCK is Ø-C

NOW D is 11 C is 12

Rock is hist Ø-D

Mungo is hot 1-C

Failed test-vendor-logic-Immed-function

was 0x8 no matter what was expected

TESTUC DATA = ADDR

ADDR 1B55 DATA - - - - 41B55
2294 — 402294 - -

B1A5 Data - 40B1AB (1st word)

7F90 47F90 (4th)

BE37 40BE37 1st

test - ALU minus K function

Ø297

{ test - uc - q/u - minus - for - for(?) - Rack

{ test - uc - cs - data - equal - address

{ 22AD circles up a 1 in (S) 21 / 22

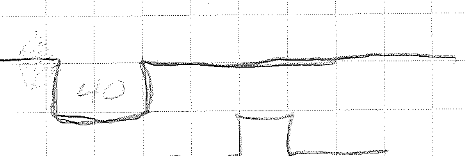
GRPA 4 — ~~GRPC~~ 0 — RSEL3
5 — 1 — " 2
6 — 2 — " 1
7 — 3 — " Ø
} 6803A

GRPA Ø - GRD<22> ✓
GRPA 1 - GBUSOUT L
GRPA 2 - ACK H
GRPA3 - RWKS<22> -

~~GRPC4 - GND~~ -
23 GRPC5 - State OK A
22 GRPC6 - WCS Reg Cntl
2 GRPPC7 - Gnd
3 GRPC8 WCS Dir<0>
21 GRPC4 WCS OE<0> L
} U313

GRPA 4 ack
5 DATA<22>
6
7

2/18 → 5 - OK
6 - Bad - neg going edge on S/15 ← only happens when "C" is ON
7 - OK
40 ns before and 40 ns wide
9 - OK (1)
11 - OK



PHONE MEMO

TO Dave Sitterfield
DATE 4/14
TIME 4:30 AM PM
FROM Jay Lebare
AREA CODE
OF
NO.
EXT.

MESSAGE

SIGNED

PHONED ☒ CALL BACK ☒ RETURNED CALL ☐ WANTS TO SEE YOU ☐ WILL CALL AGAIN ☐ WAS IN ☐ URGENT ☐

PHONE MEMO

TO Ward Datt
DATE 4/15
TIME 8:55 AM PM
FROM Dr. Collins
AREA CODE
OF
NO. 944-0389
EXT.

MESSAGE

SIGNED TV

PHONED ☐ CALL BACK ☒ RETURNED CALL ☐ WANTS TO SEE YOU ☐ WILL CALL AGAIN ☐ WAS IN ☐ URGENT ☐

C4 Pin 19 #BUS BOUT
C5

C2 pin 5 enable ABUS B
C3 pin 6 HBUS A sel /
C4 7 #BUS A sel Ø take off
C5 8 enable A take off
C6 9 #BUS B /Z
C7 11 HBUS B Ø

A Ø GBUS Block
A 1 ack in
A 2 ack out
A 3 enable HBUS B

mem Qt D 1

A 4 Reg sel Ø
A 5 sel 1
A 6
A 7 3

B 7 p 1 mem Qt D Ø

B2 pin 9 Status 22
B3 pin 10 Stat 21 } U102
B4 pin 11 Stat 20
B5 pin 12 Stat 17 ← U100
B6 Pin 5 H BUS B sel Ø7 H

A0 — G BUS OUT
A1 — ack
A2 — R sel 3
A3 — R sel 2
A4 — R sel 1
A5 — R sel 0

B6
B7 — Data out HBUS
B0 — LS 244
B5 — transceiver
B4

~~Data and ack are too close together.~~
~~400 ns apart at most.~~

Come back + look at ack.
Glitch on data<22> on rising edge of ack.